Research Product 89-06

Human Operator Simulator (HOS) IV Programmer's Guide

January 1989

Manned Systems Group
Systems Research Laboratory

U.S. Army Research Institute for the Behavioral and Social Sciences

89

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS --
2a. SECURITY CLASSIFICATION AUTHORITY --	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE --	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) --	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARI Research Product 89-06
6a. NAME OF PERFORMING ORGANIZATION Analytics, Inc. / 6b. OFFICE SYMBOL (If applicable) --	7a. NAME OF MONITORING ORGANIZATION U.S. Army Research Institute for the Behavioral and Social Sciences
6c. ADDRESS (City, State, and ZIP Code) 2500 Maryland Way Willow Grove, PA 19090	7b. ADDRESS (City, State, and ZIP Code) 5001 Eisenhower Avenue Alexandria, VA 22333-5600
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Same as 7a. / 8b. OFFICE SYMBOL (If applicable) PERI-SM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-86-C-~~19~~ 0017
8c. ADDRESS (City, State, and ZIP Code) Same as 7b.	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62717 / PROJECT NO. A790 / TASK NO. (121) 1204 / WORK UNIT ACCESSION NO. H.2

11. TITLE (Include Security Classification)

Human Operator Simulator (HOS) IV Programmer's Guide

12. PERSONAL AUTHOR(S) Harris, Regina (Analytics, Inc.); Kaplan, Jonathan (ARI); Bare, Christopher; Lavecchia, Helene, Ross, Lorna, Scolaro, Dan, and Wright, Douglas (Continued)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 1987 TO 1988	1989, January	186

16. SUPPLEMENTARY NOTATION Michael Young, Contracting Officer's Representative. Research Product prepared in cooperation with the U.S. Air Force Human Resources Laboratory, Wright-Patterson Air Force Base, Dayton, Ohio 45433.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Simulation, Modeling, Performance modeling, Human factors, Interface, HOS, Evaluation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report is a guide to maintaining and updating the source code for the Human Operator Simulator (HOS) IV, which was developed to aid in the design and evaluation of interfaces between operators or maintainers and weapon system hardware and software. HOS IV creates simulations of manned systems on an IBM-AT PC or compatible. It does this by using micromodels of basic human processes to produce both system and human performance estimates. HOS IV also includes a mechanism to aid in the creation of new micromodels.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☑ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael Young	22b TELEPHONE (Include Area Code) (513) 255-8229 / 22c. OFFICE SYMBOL AFHRL/LRG

DD Form 1473, JUN 86 *Previous editions are obsolete.*

ARI Research Product 89-06

12. PERSONAL AUTHOR(S) (Continued)

(Analytics, Inc.)

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction of the Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Technical Director

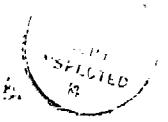
JON W. BLADES
COL, IN
Commanding

Research accomplished under contract for the Department of the Army

Analytics, Inc.

Technical review by

Christine R. Hartel
Michael Young



NOTICES

FINAL DISPOSITION: This Research Product may be destroyed when it is no longer needed. Please do not return it to the U.S. Army Research Institute for the Behavioral and Social Sciences.

NOTE: This Research Product is not to be construed as an official Department of the Army document, unless so designated by other authorized documents.

Research Product 89-06

Human Operator Simulator (HOS) IV Programmer's Guide

Regina Harris
Analytics, Inc.

Jonathan Kaplan
Army Research Institute

Christopher Bare, Helene Lavecchia, Lorna Ross, Dan Scolaro, Douglas Wright
Analytics, Inc.

Manned Systems Group
John F. Hayes, Chief

Systems Research Laboratory
Robin L. Keesee, Director

U.S. Army Research Institute for the Behavioral and Social Sciences
5001 Eisenhower Avenue, Alexandria, Virginia 22333-5600

Office, Deputy Chief of Staff for Personnel
Department of the Army

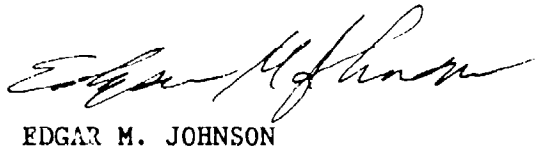
January 1989

Army Project Number
2Q 2717A790

Human Performance Effectiveness and Simulation

FOREWORD

The U.S. Army Research Institute for the Behavioral and Social Sciences (ARI) and the U.S. Air Force Human Resources Laboratory (AFHRL) have developed a simulation technique for evaluating manned systems based on their design, the performance of their operators, and the activities of their environment. This method is called the Human Operator Simulator (HOS) IV, which is a substantial alteration of the original version of HOS developed by the U.S. Navy. This method runs on an IBM AT or equivalent personal computer. HOS IV can be used at any stage of system development to model, and thus evaluate, manned developmental or nondevelopmental items. The HOS IV development is one of a number of programs with which ARI and AFHRL are advancing the state of the art of manned system design and evaluation. This specific volume is the HOS IV Programmer's Guide, which is to be used to help maintain and update source code.



EDGAR M. JOHNSON
Technical Director

EXECUTIVE SUMMARY

The Human Operator Simulator (HOS) IV allows its users to create and run computer-based simulations of manned systems in their operational environment. It also allows the development of reusable micromodels of human and hardware processes and the linking of these processes to more general system models to predict operator and overall system performance.

HOS IV runs on an IBM AT or fully equivalent machine. Written in Microsoft C, it is wholly owned by the U.S. Department of Defense. Its source code may be altered by any of its users, governmental or civilian. This report provides the documentation that such source code alteration requires. It is not a user's guide to HOS IV, although users may find it interesting and useful in understanding how HOS works.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

This document describes the Human Operator Simulator (HOS-IV) software and assumes that the reader is familiar with HOS-IV concepts and terminology as described in the HOS-IV User's Guide (Harris, et al., 1988).

1.1 Hardware Environment

HOS-IV requires an IBM PC/AT (80286 microprocessor based) or fully compatible (such as the Compaq 80286) with the following minimum configuration:

- An Enhanced Graphics (EGA) monitor,
- An Enhanced Graphics (EGA) card with 256 Kb of RAM,
- At least one 5 1/4" floppy diskette drive able to read 360 Kb formatted floppy diskettes,
- A hard disk with at least 10 Mb of available storage,
- A minimum of 640 Kb of memory (RAM),
- A minimum of 1 Mb of extended RAM that conforms to the EMS specification and functions as a RAM drive, and
- A mouse.

The optimal configuration for using HOS-IV requires the following components:

- 80287 math co-processor.
- A total of 4 Mb of RAM in the form of an additional 3.5 Mb of RAM drive. The additional RAM must conform to the EMS standard.
- Epson compatible dot matrix printer with graphics capability, such as an Epson LQ-1200.
- 40 Mb Bernoulli Box.

1.2 Software Environment

The HOS-IV software requires DOS 3.1 or higher version of the operating system. A copy of Microsoft C Version 4.0 is required. The C language was used to develop the HOS simulation software and HOS uses the C language compiler and linker. Skylights software was used to develop the user interface and therefore a copy of the Skylights software is required if any changes are made to the HOS user-computer interface.

1.3 HOS Components

The HOS-IV software is split into nine major modules; these modules perform the following functions:

- **Simulation Selection** — specifies whether a new simulation is being developed or permits selection from list of existing simulations.
- **Simulation Setup** — specifies basic simulation parameters such as simulation name, time units, simulation start time, and maximum simulation time.
- **Event Editor** — processes events including creation of new events, modification of existing events, and deletion of existing events.
- **Rule Editor** — processes rules including creation of new rules, modification of existing rules, and deletion of existing rules.
- **Action Editor** — processes actions including creation of new actions, modification of existing actions, and deletion of existing actions.
- **Object Editor** — processes objects including creation of new objects, modification of existing objects, and deletion of existing objects. The Object Editor also maintains the list of user-defined alphabetic and creates and maintains object se s.
- **Simulation Creation** — compiles the user's simulation code and links it with the HOS-IV library code.
- **Simulation Execution** — executes the user's simulation.

- **Simulation Post-Processing** — generates simulation reports.

1.4 About This Document

Section 2 of this document describes the HOS-IV user-computer interface and the use of Skylights. Section 3 presents details of the HOS-IV modules. Section 4 describes the data organization and storage. Appendix A contains detailed individual program unit descriptions organized by function; Appendix B contains a description of the error messages generated by the HOS action language translator.

2. USER INTERFACE

This section describes the user-computer interface (UCI) for HOS-IV and presents information about how the UCI was implemented using the Skylights software system.

2.1 Input Devices

The UCI input devices consist of two complementary devices — a keyboard and a mouse. The keyboard is used mainly for entering text and numbers; while the mouse is used for specifying menu options, controlling the cursor, selecting information, and specifying insertion points.

2.1.1 Keyboard

The keyboard is used to enter alphanumeric data and as an alternative to the use of the mouse to move the text cursor between input fields on dialog boxes. The keyboard consists of a standard typewriter keyboard, numeric keypad overlaid with cursor move keypad, and a set of function keys.

The numeric keypad includes keys for the numbers zero through nine arranged in an adding machine format; it also has keys for special functions (such as a minus sign, equal sign, etc.) and is used to speed entry of numeric information. The cursor movement keypad contains an up-arrow, down-arrow, right-arrow, left-arrow, home, and end keys and is used to control cursor movement within a window as an alternative to the use of the mouse. The user controls the functioning of the keypad as either a numeric pad or cursor movement pad through the use of the **NUM LOCK** key. Above the num lock key is a small red dot light. If the light is lit, then the keypad is functioning as a numeric pad, otherwise it functions as a cursor movement pad. The special function keys are used to select menu options as an alternative to the use of the mouse for experienced users with certain modules.

The following keys have special functions as described within the HOS modules indicated in parentheses:

- **Enter** (↵) is used for the following:
  1. Move the text cursor and any subsequent text to the next line (Action and Object Editor modules), and
  2. Move the text cursor to the next data entry field in the dialog window (All other modules).
- **Backspace** (←) deletes the character to the left of the text cursor. If the text cursor is positioned at the top, leftmost character in a window, subsequent depressions of the backspace key are ignored (All modules).
- **Tab** (|←) is used for the following:
  1. Insert up to five blank characters in the text depending upon the current cursor position. If the entry of the blank characters causes the width of the line to exceed the screen width, the text cursor and any subsequent text will be moved to the next line (Action Editor module), and
  2. Move the text cursor to the next input field in the dialog window (All other editors).
- ~/' key is used to generate the underscore character _ regardless of the status of the shift and/or shift lock keys.

The following keys on the cursor movement pad have special functions as described for the indicated HOS modules:

- **HOME** (above 7 key on numeric pad) is used for the following:
  1. Move the text cursor to the leftmost character in the first entry field in the dialog window (Object Editor modules) ,
  2. Move the text cursor to first screen containing text (Action Editor module) and maintain the relative position of the text cursor, and
  3. Move the text cursor to the leftmost character in a text entry box (All other modules).
- **Up-arrow** (↑)— (above 8 key on numeric pad) moves the text cursor up one line. If the current line is the top line on the page, the depression of the up arrow scrolls the page. If the current line is the first line then subsequent depressions of the up-arrow are ignored (Action Editor module).
- **PgUP** (above 9 key on numeric pad) moves the text cursor to the previous page of text and maintains the relative position of the text cursor on the page. The top line of the previous page

becomes the bottom line of the current page (Action Editor module).

- **Right-arrow (→)** (above 6 key on numeric pad) moves the text cursor one character to the right (All modules).
- **Left-arrow (←)** (above 4 key on numeric pad) moves the text cursor one character to the left (All modules).
- **END** (above 1 key on numeric pad) is used for the following:
  1. Move the text cursor to the rightmost character in the last entry field in the dialog window (Object Editor module),
  2. Move the text cursor to last screen containing text and maintain the relative position of the cursor (Action Editor module), and
  3. Move the text cursor to the right most character in a text entry box (All other modules).
- **PgDN** (above 3 key on numeric pad) moves the text cursor to the next page of text and maintains the relative position of the text cursor on the page. The bottom line of the previous page becomes the top line of the current page (Action Editor module).
- **Down-arrow (↓)**— (above 2 key on numeric pad) moves the text cursor down one line. If the current line is the last line of the page, the down-arrow will scroll the text down one line. If the current line is the last text line, subsequent depressions of the down-arrow will be ignored (Action Editor module).

The following functions keys are used for the indicated processes only within the Action Editor:

- F1 — Begin text marking for cut/copy operation.
- F2 — End text marking for cut/copy operation.
- F3 — Cut text.
- F4 — Copy text.
- F5 — Paste text.
- F6 — Clear text.

2.1.2 Mouse

The mouse is used as a pointing device to select commands from menus, to control cursor (pointer) movement and to manage file scrolling. In HOS-IV, the standard pointer is an arrow (↖). Every move of the mouse moves the

pointer in exactly the same way. The following terms describe various actions associated with mouse utilization:

- **Clicking** — positioning the pointer with the mouse, briefly pressing, and releasing the mouse button without moving the mouse.
- **Pressing** — positioning the pointer by holding down the mouse button without moving the mouse.
- **Dragging** — positioning the pointer with the mouse, holding down the mouse button, moving the mouse to a new position, and then releasing the button.

These terms will be used in subsequent sections to describe user interactions with HOS-IV.

2.2 Screen Layout and Components

This section describes how information is arranged on the display and how the user interacts with the particular component. The screen is arranged into three main areas:

1. The **title bar** which is always on the top line of the display as described in Section 2.2.1;
2. The **menu bar** which is always the second line of the display as described in Section 2.2.2; and
3. The **HOS window** which occupies the remainder of the screen. The HOS window is used to conduct a dialog with the user. It will either display information to the user or display an input form for the user to supply the information HOS requires. The contents vary dependent upon the current function. The windows are described in Sections 2.2.3 through 2.2.7.

Within the HOS window, a variety of components have been developed for the user to specify particular simulation data items or supplying additional information required before a system command can be processed. These components include:

1. **List Selection Box** — a scrollable list of available items for the user to select from as described in Section 2.2.8;
2. **List Viewing Box** — a scrollable list of currently defined terms for the user to view as described in Section 2.2.9;

3. **Pushbutton** — a distinct area of the screen that is used to specify actions as described in Section 2.2.10;
4. **Click Boxes** — a box containing the range of numeric values that can be modified by the user via mouse clicks as described in Section 2.2.11;
5. **Text Entry Boxes** — a rectangular box which allows the user to enter textual data (numeric or alphanumeric) with the size of the box indicating the maximum number of characters permitted as described in Section 2.2.12;
6. **Scroll Bar** — a rectangular box that is used to modify the current view of a window as described in Section 2.2.13; and
7. **Labels** — text descriptions used to indicate the type of information to be entered by the user as described in Section 2.2.14.

HOS uses two distinct cursors to represent the focus of attention for the user and to point to a precise point on the screen. The **mouse cursor** is controlled by the mouse and always represents the last mouse screen location. When the user moves the mouse, the mouse cursor moves proportionately. The mouse cursor is described in Section 2.2.15. Additionally, a **keyboard cursor** represents the location where any keyboard actions will occur and is described in Section 2.2.16.

In subsequent sections the following terminology is used to describe various aspects of the screen components:

- Location — indicates where the component is placed on the display. In some cases, the exact location will vary depending upon the contents of the screen;
- Background Color — indicates the color to be used for the screen background upon which text and/or graphics will appear;
- Text Color — indicates the color to be used for all alphanumerics and text symbols;
- Graphics Color — indicates the color to be used for graphics symbols;
- Border — indicates the color to be used for the line border enclosing a particular element of the screen;
- Characters — indicates the case to be used for text, e.g., all upper case, initial upper case, etc.; and

- User Action — describes how the user will interact with the particular screen component.

2.2.1 Title Bar

The title bar presents information about the current HOS function to the user; the user does not enter any information. The title bar is continuously displayed and is illustrated in Figure 2-1. The title bar is split into three areas:

1) Current activity on left side of line, left justified with initial caps, if required. (Currently used by Action Editor.)
2) Function name centered in middle of line in uppercase white letters on black background. (Required for all HOS modules.)
3) Status information on right side of line with initial caps, if required. (Currently used by Action Editor to display line count and column position.)

Location:	Top line of screen.
Background Color:	White
Text Color:	Black
Graphics Color:	None
Border:	None
Characters:	Varies depending upon area.
User Action:	Information only, no user response permitted in the title bar area of the screen.

2.2.2 Menu Bar

The menu bar contains a list of the menu titles of the primary options that are available for the current HOS function. The menu bar is continuously displayed on the screen and is illustrated in Figure 2-1. The last two menu options are always User Aids and Exit. Each menu title is separated from other menu titles by one leading and two trailing spaces.

Location:	Second line on screen.
Background Color:	Blue
Text Color:	White
Graphics Color:	None

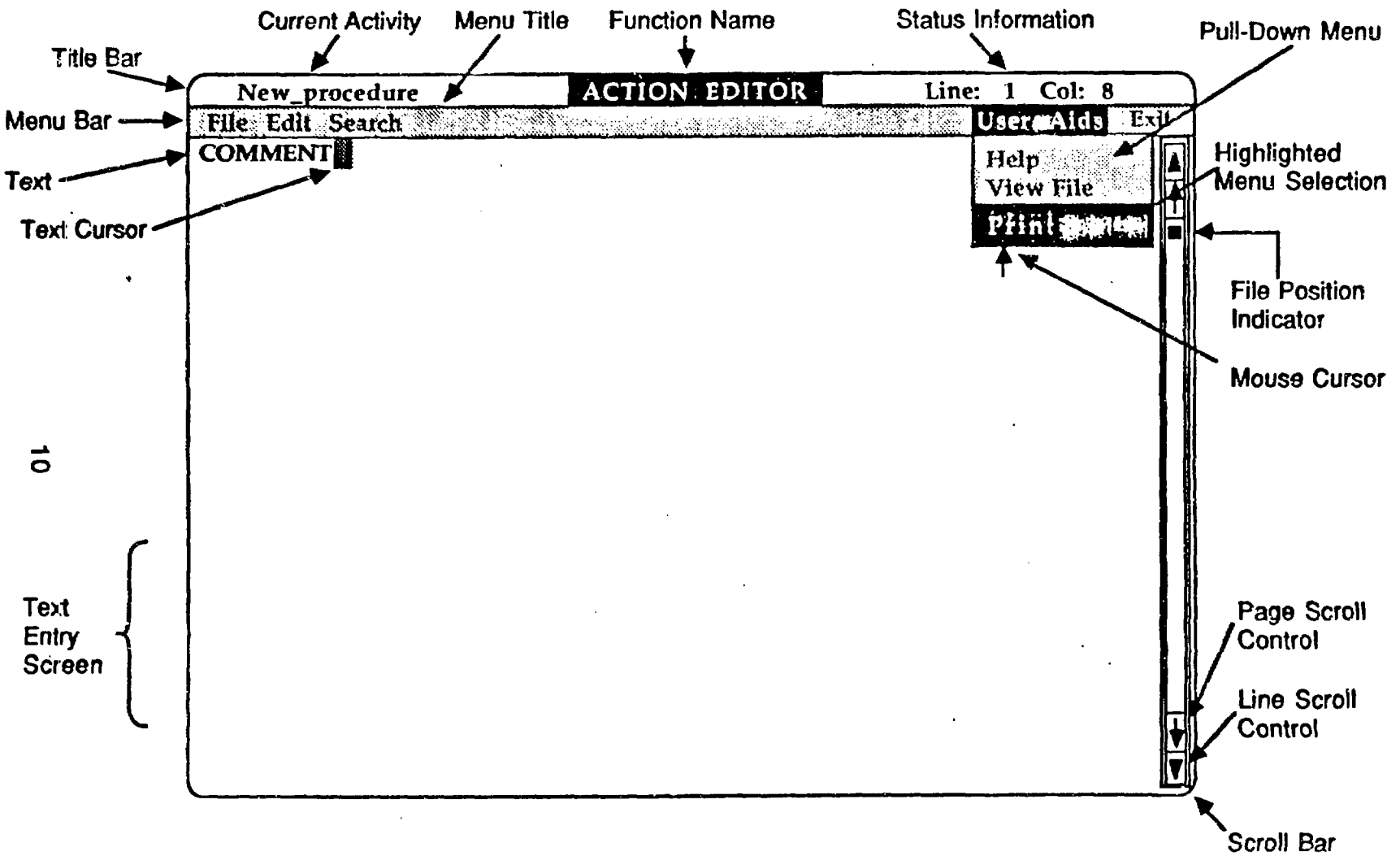


Figure 2-1. HOS Screen Components

Border: None

Characters: Initial upper case only.

User Action: To select an option from the menu bar, the mouse is used to position the mouse cursor anywhere on the desired menu title. Without moving the mouse, any mouse button is pressed and held (clicking). Once the mouse button is depressed, the selected menu title will be shown in reverse video (white background and blue foreground) and a box containing the available commands (pull-down menu) will appear immediately beneath it in a separate window. The pull-down menu will disappear as soon as the mouse button is released. In order to view all the pull-down menus, the user can drag the mouse across the menu bar and, as each menu title is selected, the accompanying pull-down menu will be displayed.

2.2.3 Pull-Down Menus

The pull-down menu is a separate rectangular window displayed beneath the menu bar containing the list of commands available for a particular menu title on the menu bar. It is illustrated in Figure 2-1. It is displayed only from the time the mouse button is held down and the mouse arrow is dragged down through the menu options until the mouse button is released. The pull-down menu window may obscure the previous contents of the screen while it is active.

Location: A separate rectangular window whose top is immediately beneath the menu bar and upper left corner is aligned with the selected menu title. The menu text is indented two spaces to the left of the selected menu title. The window is one space wider than the title of the longest command title.

Background Color: Blue

Text Color: White

Graphics Color: None

Border: None

Characters: Initial upper case only.

User Action: To choose one of the listed commands in the pull-down menu, the mouse is used to move the mouse pointer to the displayed menu title on the menu bar. While the mouse button is held down, the mouse is used to move the mouse pointer to the desired command (dragging). When the mouse pointer is located over the selected command, the mouse button is released. As the mouse pointer moves to each command line, the currently selected command is highlighted in reverse video (white foreground and blue background). The command that is highlighted when the mouse button is released is invoked and the pull-down menu disappears. If the mouse cursor is relocated within the menu bar line and the mouse button released, no action will occur and the pull-down menu will disappear. Similarly, if the mouse cursor is dragged outside of the pull-down menu window and released, the pull-down menu will disappear and no command will be chosen.

2.2.4 Message Windows

A message window presents informative messages about the current system action requesting the user to indicate subsequent actions that should occur or be cancelled. The options available to the user are displayed as pushbuttons and one of the pushbuttons must contain a CANCEL option that permits the user to cancel the current request and resume the previous activity. The message window is illustrated in Figure 2-2.

Location: A separate rectangular window that is located in the workspace beneath the menu bar.

Background Color: Blue

Text Color: White

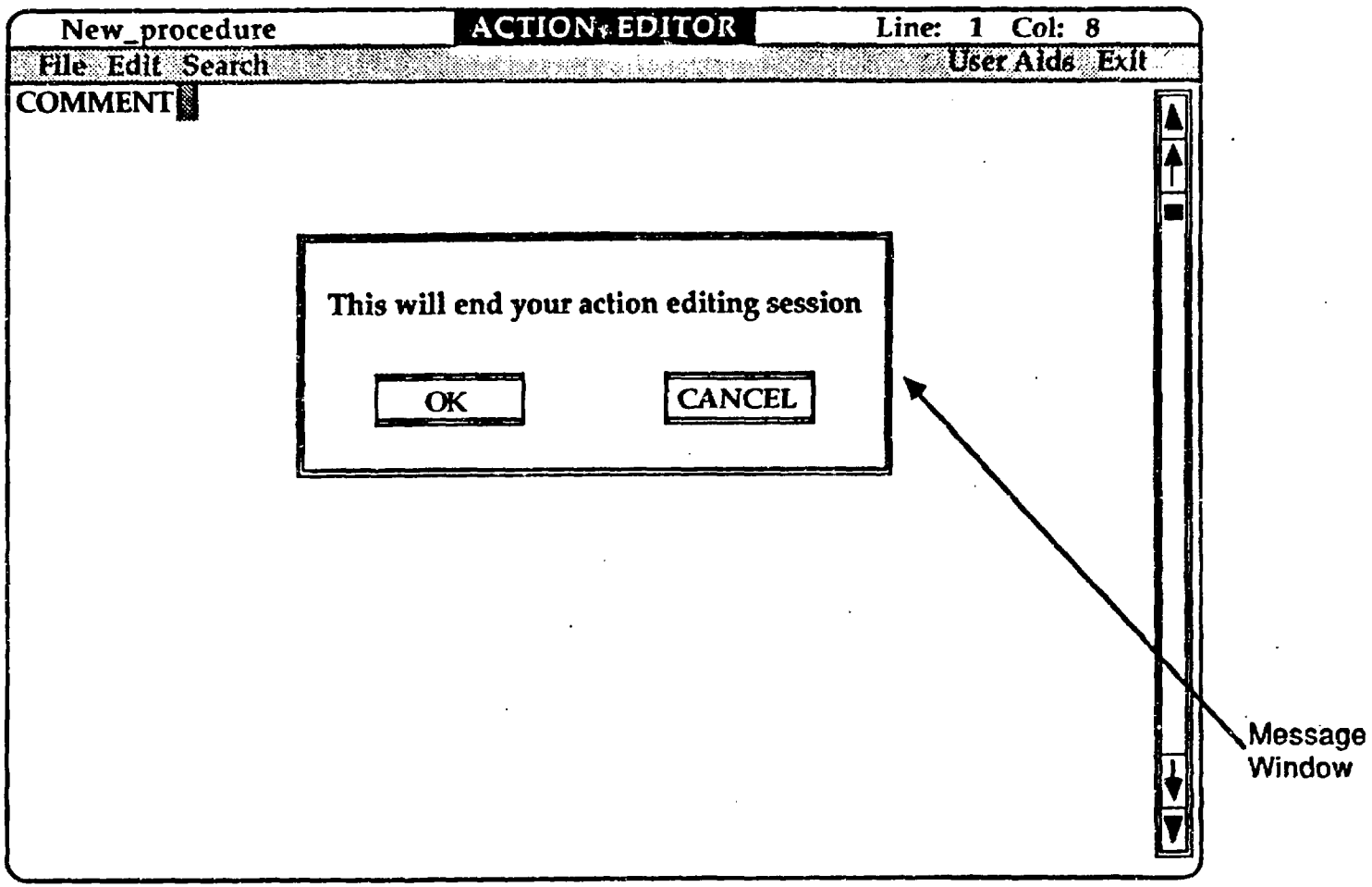


Figure 2-2. HOS Message Window Screen

Graphics Color: White
Border: Double Line
Characters: Message displayed in sentence format with initial caps centered in window. Pushbutton labels follow pushbutton format.
User Action: The mouse is used to move the mouse cursor to the pushbutton containing the desired action and depressed.

2.2.5 Dialog Windows

Dialog windows allow the user to enter necessary information in pre-defined fields consisting of pushbuttons, click boxes, check boxes, text entry boxes, labels, and scroll bars. An example is shown in Figure 2-3.

Location: A separate rectangular window that is located in the workspace beneath the menu bar.
Background Color: White
Text Color: Black
Graphics Color: Blue
Foreground Color: Blue
Border: Double line
Characters: Title in upper case centered in top line of window.
User Action: The text cursor (cyan background, black foreground) is initially placed in the beginning of the first text entry box for the user to enter the indicated information. When the entry is completed, the user can depress the return key to move the text cursor to the next item in the sequence or, alternatively, use the mouse to move the mouse pointer to the desired field and click to obtain the text cursor in the desired location. In addition, the tab key can be used to move forward to the next text entry field.

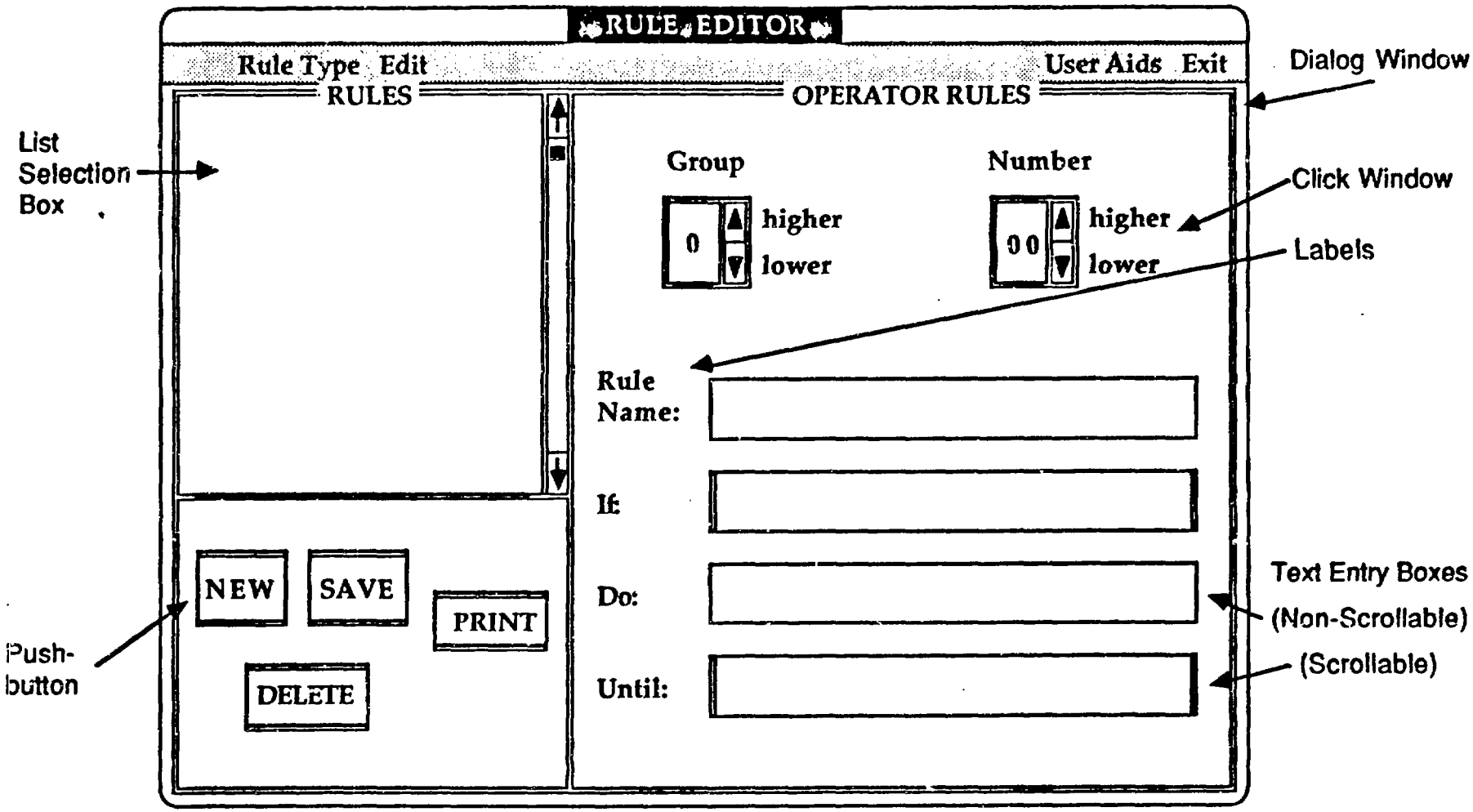


Figure 2-3. HOS Dialog Window & Screen Components

2.2.6 Information Windows

An information window presents messages to the user about current system activity and is illustrated in Figure 2-4. An information window differs from a message window in that the user cannot make any responses.

Location:	A separate rectangular window that is located in the workspace beneath the menu bar.
Background Color:	Blue
Text Color:	White
Graphics Color:	White
Border:	Double line
Characters:	Message displayed in sentence format with initial caps centered in window.
User Action:	Information only, no user response permitted in the information window area of the screen.

2.2.7 Text Entry Screens

A text entry screen is a user scrollable window in which the user can enter textual/numerical information in a free-format. A scroll bar is displayed on the right side of the window and is illustrated in Figure 2-1.

Location:	A separate rectangular window that is located in the workspace beneath the menu bar.
Background Color:	White
Text Color:	Black
Graphics Color:	Blue
Border:	Double Line
Characters:	Title in Upper Case; contents dependent upon user entries.

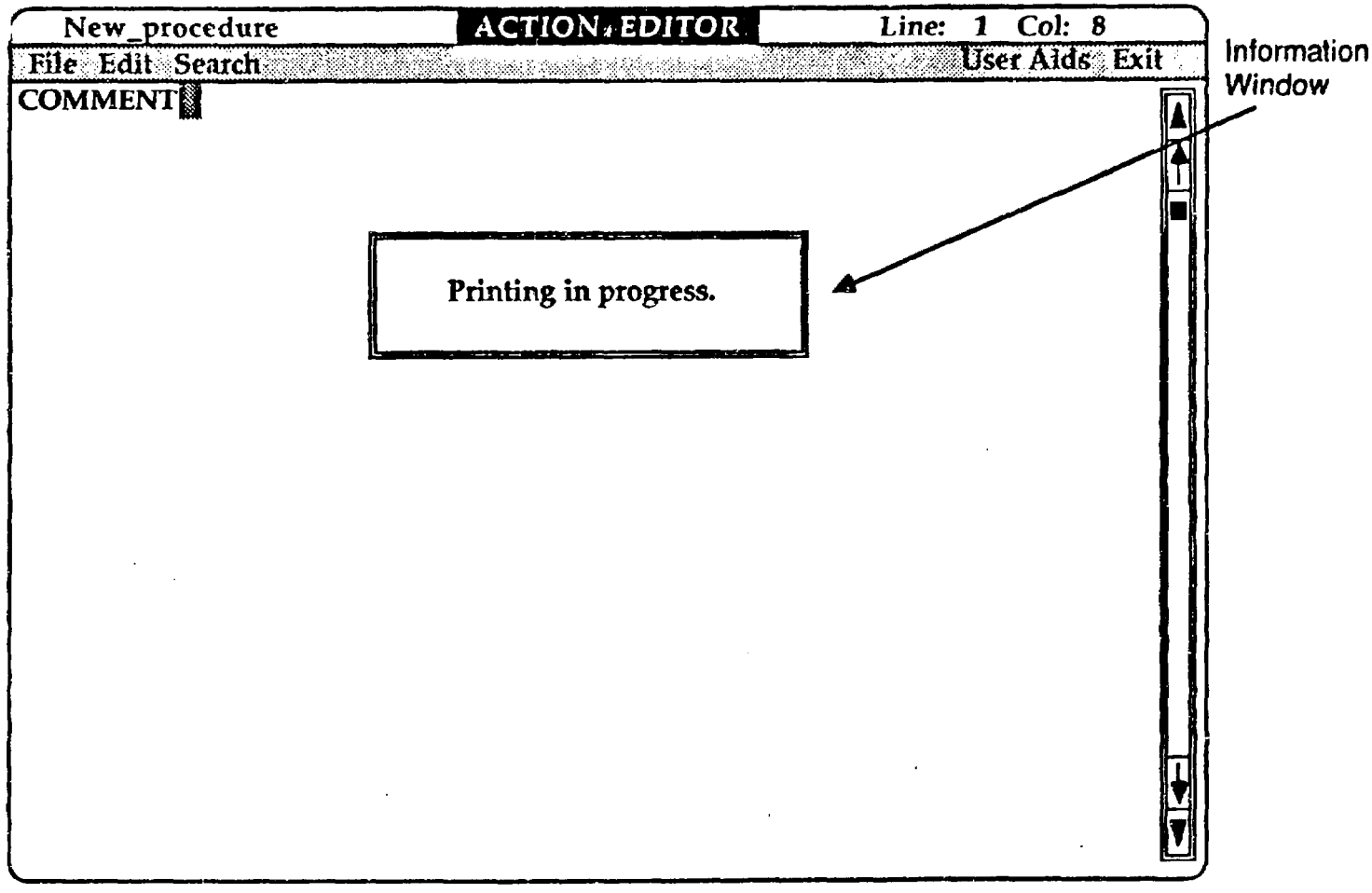


Figure 2-4. HOS Information Window Screen

User Action: The rectangular text cursor is initially placed at the upper left corner of the window. The user can use the mouse or arrow keys (right, left, up, down, home, page up, and page down) to position the text cursor to the location where the next keyboard entry is to be placed. All key strokes are inserted at the current location of the text cursor. If the entry causes the length of the current line to exceed the display width, all text following the previous delimiter (space) is moved to the next line. The depression of a carriage return moves the text cursor and any text after it to the next line. The home key moves the text cursor to the first window of text; the end key moves the text cursor to the last window containing text.

2.2.8 List Selection Box

A list selection box presents a list of all items available to the user for the current function, e.g., list of rules for the Rule Editor, actions for the Action Editor, objects for the Object Editor, etc. It requires a scroll bar on the right side of the window to be used to alter the viewing area of the window. The list box window may obscure the previous contents of the screen while it is active. It is illustrated in Figure 2-3.

Location: A separate rectangular window that is located in the workspace beneath the menu bar.
Background Color: White
Text Color: Black
Graphics Color: Blue
Border: Double Line
Characters: Title in upper case in center of top line of window; pushbutton labels follow pushbutton format; remainder dependent upon user inputs.
User Action: The mouse pointer is initially located on the first item in the window. The mouse is used to move the

mouse cursor so as to point to the name of the item to be selected. The currently selected item is shown in reverse video. If the desired item is not currently displayed within the window, the user can move the mouse cursor to the red triangles located at either end of the scroll bar and then depress the mouse button. Each click on the red triangle will display the next set of items in the window. If the user clicks on the up (down) triangle and the pointer is already located at the first (last) item, the contents of the screen will remain identical. Once the desired item is selected, the mouse cursor must be moved to the appropriate pushbutton to invoke the desired action.

2.2.9 List Viewing Box

The list viewing box displays a list of all defined items for the user to view, e.g., list of alphabetic for the Object Editor, etc. It requires a scroll bar on the right side of the window to be used to alter the viewing area of the window. The list viewing box window may obscure the previous contents of the screen while it is active. It is illustrated in Figure 2-5.

Location:	A separate rectangular window that is located in the workspace beneath the menu bar.
Background Color:	White
Text Color:	Black
Graphics Color:	Blue
Border:	Double Line
Characters:	Title in upper case in center of top line of window; pushbutton labels follow pushbutton format; remainder dependent upon user inputs.
User Action:	The mouse pointer is initially located on the first item in the window. If the desired item is not currently displayed within the window, the user can move the mouse cursor to the red triangles located at either end of the scroll bar and then depress the mouse

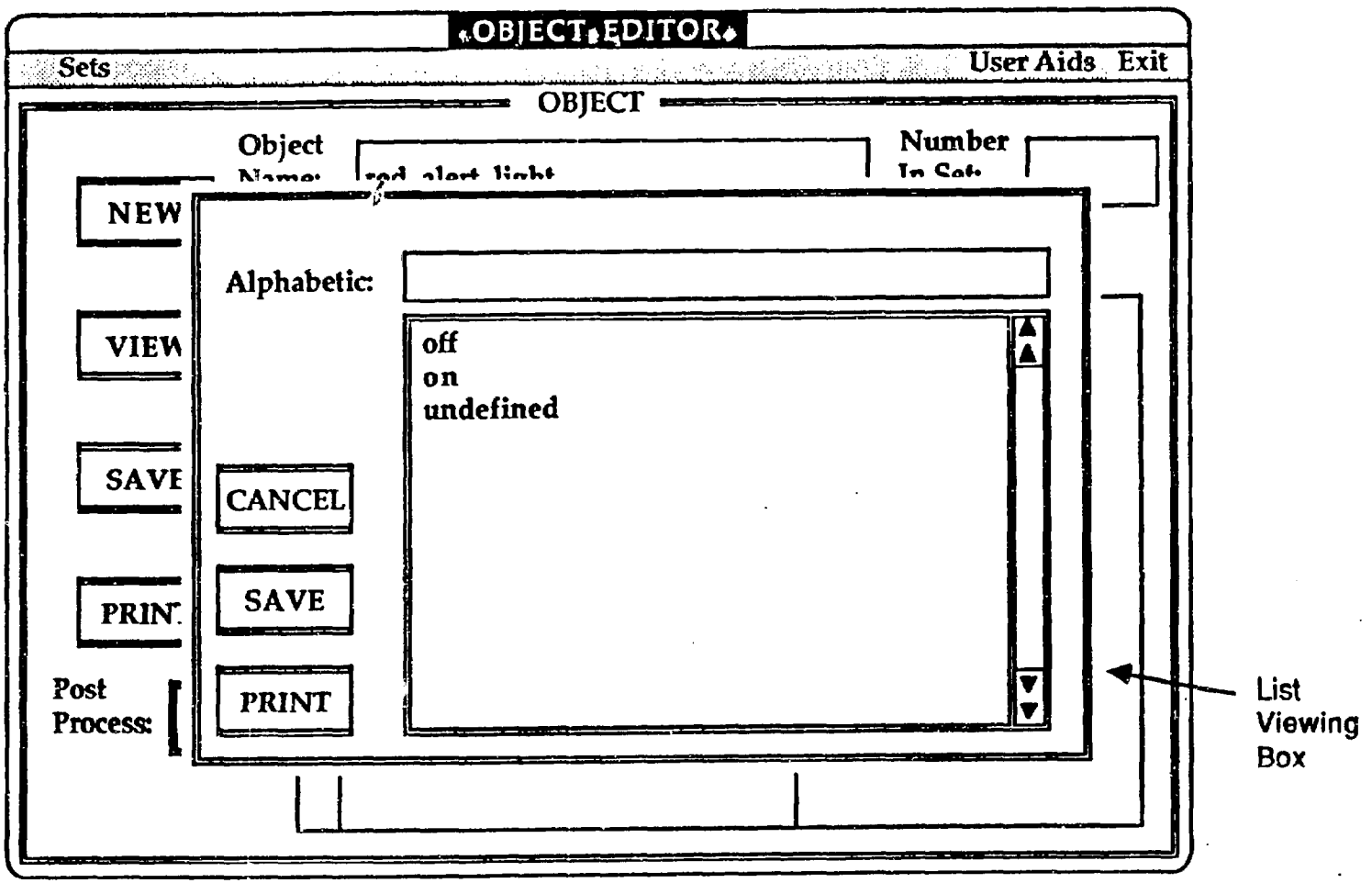


Figure 2-5. List Viewing Box

button. Each click on the red triangle will display the next set of items in the window. If the user clicks on the up (down) triangle and the pointer is already located at the first (last) item, the contents of the screen will remain identical. Once the viewing of the defined items is complete, the mouse cursor must be moved to the appropriate pushbutton to invoke the desired action.

2.2.10 Push Buttons

Pushbuttons perform instantaneous action as described by the text label with a mouse click anywhere within the button area. They are illustrated in Figure 2-3.

Location:	A separate rectangular window that is located in the workspace beneath the menu bar.
Background Color:	Assumes background color of item beneath.
Text Color:	Red
Graphics Color:	Assumes foreground color of item beneath.
Border:	Rectangular box with double line on top and bottom; single line on left and right. It is sized so that there are at least two leading and trailing spaces around the pushbutton legend.
Characters:	Upper case button label centered in box.
User Action:	The mouse is used to move the mouse cursor so that the pointer is located anywhere within the rectangular area and then a mouse button is clicked. Once any mouse button is depressed, the button label is shown in reverse video (red background and white foreground) and the indicated function immediate voked.

2.2.11 Click Boxes

Click boxes are used to specify numeric values and are illustrated in Figure 2-3. It requires a scroll bar on the right side of the window to be used to alter the numeric value currently displayed in the window.

Location: Varies but always within a dialog window.
Background Color: White
Text Color: Black
Graphics Color: Blue
Border: Rectangle with double line border.
Characters: Title in upper case in center of top line of window;.
User Action: The user can modify the currently displayed value by:

1. Moving the mouse pointer to the red triangle located above the box to increase the number shown in the box by 1 each time a mouse button is depressed, or by
2. Moving the mouse pointer to the red triangle located beneath the box to decrease the number shown in the box by 1 each time a mouse button is depressed.

2.2.12 Text Entry Boxes

Text entry boxes are fields where textual or numerical data are entered and are illustrated in Figure 2-3.

Location: Varies but always within a dialog window.
Background Color: White
Text Color: Black
Graphics Color: Blue
Border: Rectangular box drawn with single blue line. If the entry in the box can be bigger than the box size, a double blue line is placed on the left and right to indicate that the user can scroll right and left within this box.

Characters: Label with initial caps terminated with a colon to the left of the box; entries in box are based upon user actions.

User Action: The rectangular text cursor is initially placed at the left side of the box. The user can use the mouse or arrow keys (right and left) to position the text cursor to the location where the next keyboard entry is to be placed. All keyboard strokes are inserted at the current location of the text cursor. If the entry causes the length of the current line to exceed the display width, either of the following will occur:

1. If the box is the exact size of the permitted entry (i.e., the right and left side are single lines), a beep will be sounded and future keyboard entries (except backspace and delete) will be ignored until a non-text key is depressed or
2. If the entry can be larger than the box (i.e., the right and left sides are double lines), the text will be scrolled to the left as additional keys are depressed until the maximum field size is reached.

The left and right arrow keys move the cursor one space in the indicated direction within the text entry box. The home key moves the text cursor to the first character in the box; the end key moves the text cursor to the last character in the box.

2.2.13 Scroll Bar

Scroll bars are used to change which part of a list of items (list window) or contents of a file (text entry window) is shown the window. Double red scroll arrows are used at the top and bottom of the scroll bar rectangle to indicate the direction the viewing area is to be moved. The top arrow (▲) is used to scroll up one line at a time; the down arrow (▼) is used to scroll down one line at a time.

The second up arrow (⇑) is used to scroll up one page at a time; likewise the top down arrow (⇓) is used to scroll down one page at a time. The scroll bar is illustrated in Figure 2-1.

Location:	Rectangular box shown on right side of text entry and list boxes.
Background Color:	White
Text Color:	None
Graphics Color:	Blue
Border:	Double line.
Characters:	Graphics characters of ▼ and ▲.
User Action:	The user uses the mouse to position the mouse cursor at the desired scroll arrow and clicks to alter the contents of the window. The content of the window is moved in the opposite direction from the arrow. For example, when the user clicks the top scroll arrow, the contents move down, bringing the view closer to the top of the list or document. Each click of the single arrow moves the window contents one line in the chosen direction; each click of the double arrow moves the window contents one page in the chosen direction. Continuous depression of the mouse results in continuous movement in the chosen direction. Once the top or bottom of the window contents is reached, depression of the scroll arrows in that direction are ignored.

2.2.14 Labels

A label is an alphanumeric description of the information to be entered for a component of a dialog window and is illustrated in Figure 2-3.

Location:	Varies
Background Color:	White
Text Color:	Blue
Graphics Color:	None

Border: None
Characters: Initial upper case and terminated with a colon.
User Action: None

2.2.15 Mouse Cursor

The mouse cursor is a pointing device used to select commands from menus, to control cursor movement, and to manage file scrolling. It is illustrated in Figure 2-1.

Location: Varies
Background Color: Assumes background color of object beneath.
Text Color: None.
Graphics Color: Assumes foreground color of object beneath.
Border: None
Characters: Arrow (↖)
User Action: The user moves the mouse cursor to the desired location by moving the mouse in the desired direction. Every mouse movement moves the mouse cursor in exactly the same way. The following mouse actions can be performed:

- **Clicking** — positioning the mouse cursor with the mouse, briefly pressing and releasing the mouse button without moving the mouse.
- **Pressing** — positioning the mouse cursor with the mouse, holding down the mouse button without moving the mouse.
- **Dragging** — positioning the mouse cursor with the mouse, holding down the mouse button, moving the mouse to a new position, then releasing the button

2.2.16 Text Cursor

The text cursor indicates where next keyboard stroke will be entered and is illustrated in Figure 2-1.

Location:	Varies
Background Color:	Cyan
Text Color:	Black
Graphics Color:	Assumes foreground color of object beneath.
Border:	None
Characters:	Rectangle the size of a single character ( ).
User Action:	The user can use the arrow keys or the mouse to indicate where the text cursor should be positioned. Each subsequent keyboard stroke will be inserted at the location of the text cursor.

2.3 Skylights/GX Usage

Skylights provides two window editors — one alphanumeric and the other graphic for building windows and defining "touch zones." The graphics editor was not used in this implementation of HOS-IV. The user interface was written using alphanumeric windows because the speed of execution is much greater. Touch zones are areas on the screen that cause the associated function defined in the window editor ("demon") to be executed when the mouse cursor moves over them or the user clicks on them. This allows the development of an event driven user interface that is necessary for a menu/mouse driven program. The alphanumeric window editors were used in conjunction with a extensive library of C callable routines to develop the user interface described in Section 2.

A key feature of Skylights is the ability to interactively define various screen components and store them in a library. With other packages and tool kits, common front end structures and visuals, such as windows, menus, icons, boxes, and screen captions, are often implemented dynamically in the application program. Skylights uses a different approach. Most of the necessary structures and visuals are created interactively with the Skylights

editor and saved in separate window catalog files. Low-level functions for screen handling, drawing boxes, color definition, etc. are available.

The Skylights editor is used to create windows. A window is a fragment of the screen which has a picture and imbedded active areas (touch zones), menus, icons, etc. Windows are saved in a file called a window catalog. The defined windows are loaded into an application at run time using library functions calls.

Using a pointing device, a window is defined, character graphics are drawn to create the screen display and shape, and touch zones are specified as well as specifying how the application should respond to the touch event. In addition, optional properties such as audio feedback, video feedback, and the name of the routine responsible for processing touch events in the zone can be specified. The run time libraries contain window management functions, and a touch events handler, as well as various screen, keyboard, and speaker handling functions.

3. PROGRAM ORGANIZATION

This section presents a high-level functional description of each of the main HOS system processes with the overall system organization described in Section 3.1. Details of the individual program units for each module are presented in Appendix A.

3.1 System flowchart

The HOS-IV software is split into the following major system modules as described below:

- **HOS-IV**: high-level system controller that manages the spawning processes for lower-level modules.
- **SELECT:** determines which of the existing simulations is to be used during the HOS simulation session and permits the user to define a new simulation.
- **SETUP:** determines basic simulation parameters such as simulation name, time units, simulation start time, and maximum simulation time based upon.
- **EVE_EDIT**: maintains the event data base for a simulation including creation of new events, modification of existing events, and deletion of existing events.
- **RULEEDIT**: maintains the rule data base for a simulation creation of new rules, modification of existing rules, and deletion of existing rules.
- **ACTEDIT**: maintains the action library and processes user actions to create new actions, modify existing actions, and deletes existing actions. In addition, ACTEDIT invokes the action translator (HAL) that translates the action into C code and determines if the action contains any errors.
- **EDIT_OBJ**: maintains the object and alphabetic libraries and processes user actions to create new objects/alphabetics, modify existing objects/alphabetics, and delete existing objects/alphabetics.
- **HAL**: evaluates the syntax of an action and if no errors are detected translates the action into HOS C code.

- **SIMLINK:** integrates all the user's events, rules, actions, and objects into files that are compiled using the Microsoft C compiler and builds the simulation executable file.
- **SIMRUN:** executes the user's simulation and creates files for post-processing.
- **RESULTS:** generates simulation reports.

The organization of the HOS software is presented in Figure 3-1 with the left to right progression showing the sequence in which the modules must be invoked in order to properly construct and execute a simulation. The SELECT module must always be selected at the beginning of each HOS session in order for the user to specify which simulation is to be used during the session. The sequence of the remainder of the modules is user-driven. All of the modules use the User-Computer Interface (UCI) described in Section 2.

Certain conventions are used throughout the HOS modules. All names are a maximum of 28 alphanumeric characters (a-z, 0-9). The first letter of each name must be an alphabetic (a-z). The only special character that can be included is an underscore (_). HOS is also case insensitive, that is upper and lower case characters can be used interchangeably. All names are converted by HOS into lower case for internal use. Examples of valid names are myname, my_sim_variable_name, and abc123456.

Examples of invalid names are:

a$	Contains an invalid special character ($)
a-b	Contains an invalid special character (-)
1qwerty	First letter is not alphabetic

3.2 HOS-IV Modules

3.2.1 HOS-IV

HOS-IV is the main module in the system and is invoked when the system is initiated. It is the top level module that controls interactions with all other modules in the HOS system. Whenever the user terminates one of the

lower level modules, control is returned to HOS-IV for the selection of the next module or to terminate the system.

3.2.1.1 Description

HOS-IV controls user access to all of the HOS modules. There are nine commands which can be executed from the main screen of HOS-IV — Select Simulation, Setup Simulation, Edit Events, Edit Rules, Edit Actions, Edit Objects, Create Simulation, Run Simulation, and View Results. A pushbutton was created for each function and the user invokes the desired function by moving the mouse over the title of the selected pushbutton and clicking the mouse. HOS-IV automatically initiates the Select Simulation command on start up (see section 3.2.2). After the user has successfully completed the simulation select step, any of the other functions may be chosen. A high-level functional diagram of HOS-IV is illustrated in Figure 3-1.

3.2.1.2 HOS-IV Screens

The HOS-IV screen contains a diagram of pushbuttons and arrows indicating the functional flow of the steps involved in developing, executing, and analyzing a HOS simulation.

Action Editor Title Bar. The title bar of HOS-IV contains the words 'HOS-IV' as the function name. HOS-IV does not use the current activity or status information areas of the title bar.

HOS-IV Menu Bar. HOS-IV contains the following menu options on the menu bar as illustrated in Figure 3-2:

- **User Aids** - provides the capabilities to view help files.
- **Exit** - terminates HOS-IV and returns the user to the operating system.

The **User Aids** pull down menu contains the **Help** commands that allow the user to obtain help windows about the HOS-IV module.

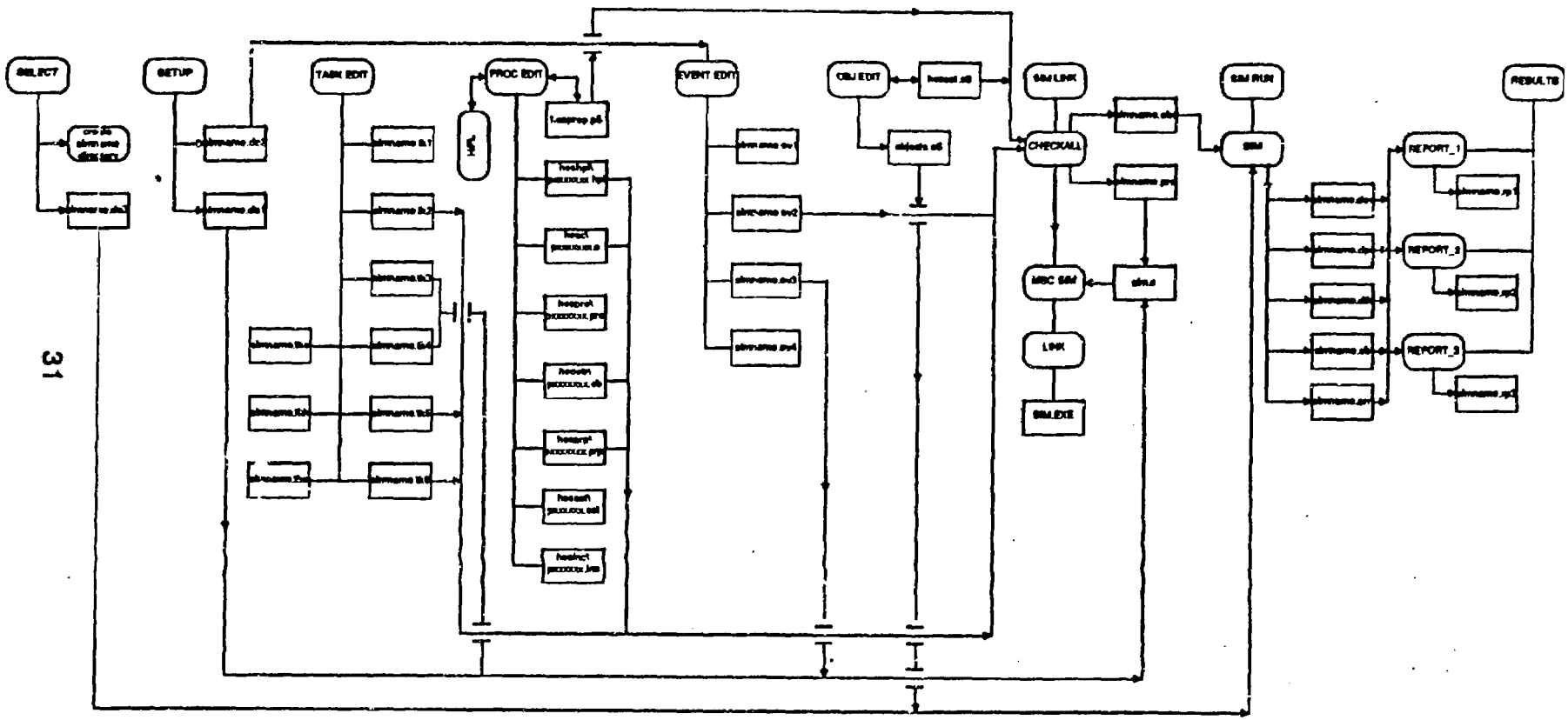


Figure 3-1. HOS-IV Software Flow Chart

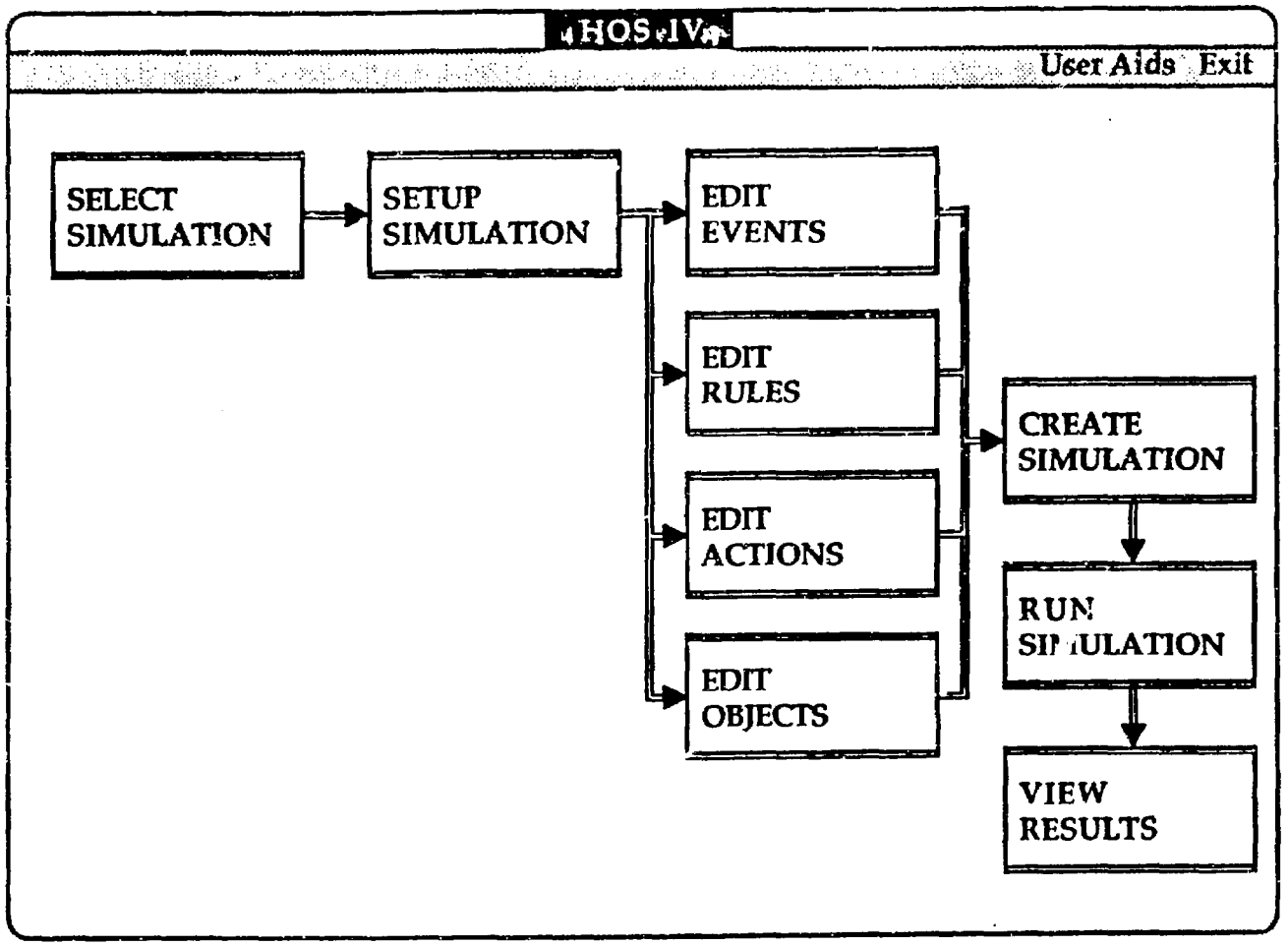


Figure 3-2. HOS-IV Main Screen

HOS-IV Main Screen. The HOS-IV main screen is illustrated in Figure 3-2 and contains the following pushbuttons:

- **Select Simulation** — calls the select simulation routine
- **Setup Simulation** — spawns the setup simulation module
- **Edit Event** — spawns the Event Editor
- **Edit Rule** — spawns the Rule Editor
- **Edit Action** — spawns the Action Editor
- **Edit Object** — spawns the Object Editor
- **Create Simulation** — spawns the simlink module
- **Run Simulation** — spawns the simulation
- **View Results** — spawns the view results module

3.2.1.3 Maintenance Procedures

HOS-IV source code (HOS-IV.c and hos_menu.c) is compiled using Microsoft C version 4.0 with the large memory model switch (/AL). It requires the HOS-IV, SKYL, CT, and TE libraries in addition to the standard C libraries for linking.

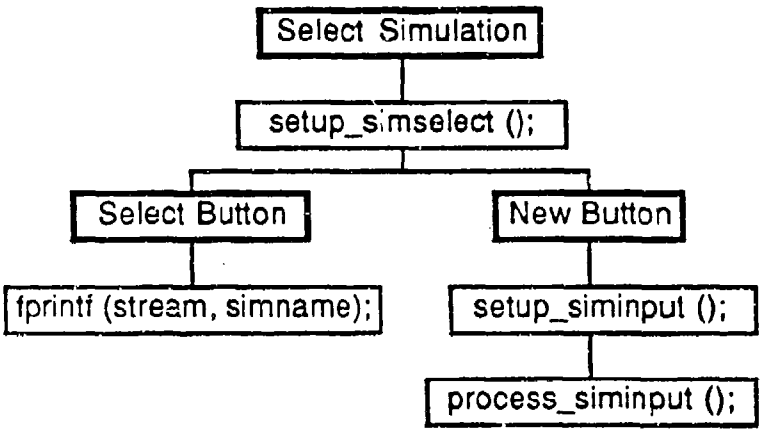
3.2.2 SELECT — Select Simulation

The SELECT module determines the simulation to be processed and permits the user to define new simulations.

3.2.2.1 Description

The SELECT module is automatically executed whenever HOS is first executed so that the user is forced to select a simulation before any other HOS functions can be invoked. In addition, the user can depress the SELECT pushbutton whenever the HOS main menu is displayed in order to switch to a different simulation. If any simulations exist, SELECT displays a list selection box which allows the user to scroll through the list of currently defined simulations and select one of the listed simulations. The user can name a new simulation or cancel the select function. If no simulations exist, the user can only enter a new simulation name or cancel. Whenever the user cancels

SELECT without selecting a simulation, an informative message is displayed indicating that the HOS session cannot continue until a simulation is selected. The response to this message is to exit (terminate the HOS session) or continue (return to SELECT). The name of a simulation can contain a maximum of eight alphanumeric characters. The simulation name cannot contain any special symbols except underscore (_). Examples of valid simulation names are my_sim, teampack,etc. A high-level functional diagram of the SELECT process is illustrated below.



3.2.2.2 SELECT Screens

The SELECT simulation user windows overlay the main HOS simulation flow screen and is illustrated in Figure 3-3.

Title Bar. The title bar of SELECT contains the words 'HOS-IV' as the function name. SELECT does not use the current activity or status information areas of the title bar.

SELECT Menu Bar. SELECT contains the following menu options on the menu bar as illustrated in Figure 3-3:

- **User Aids** — provides the capability to print the simulation names and obtain help messages; and
- **Exit** — terminates SELECT and returns the user to the HOS-IV screen.

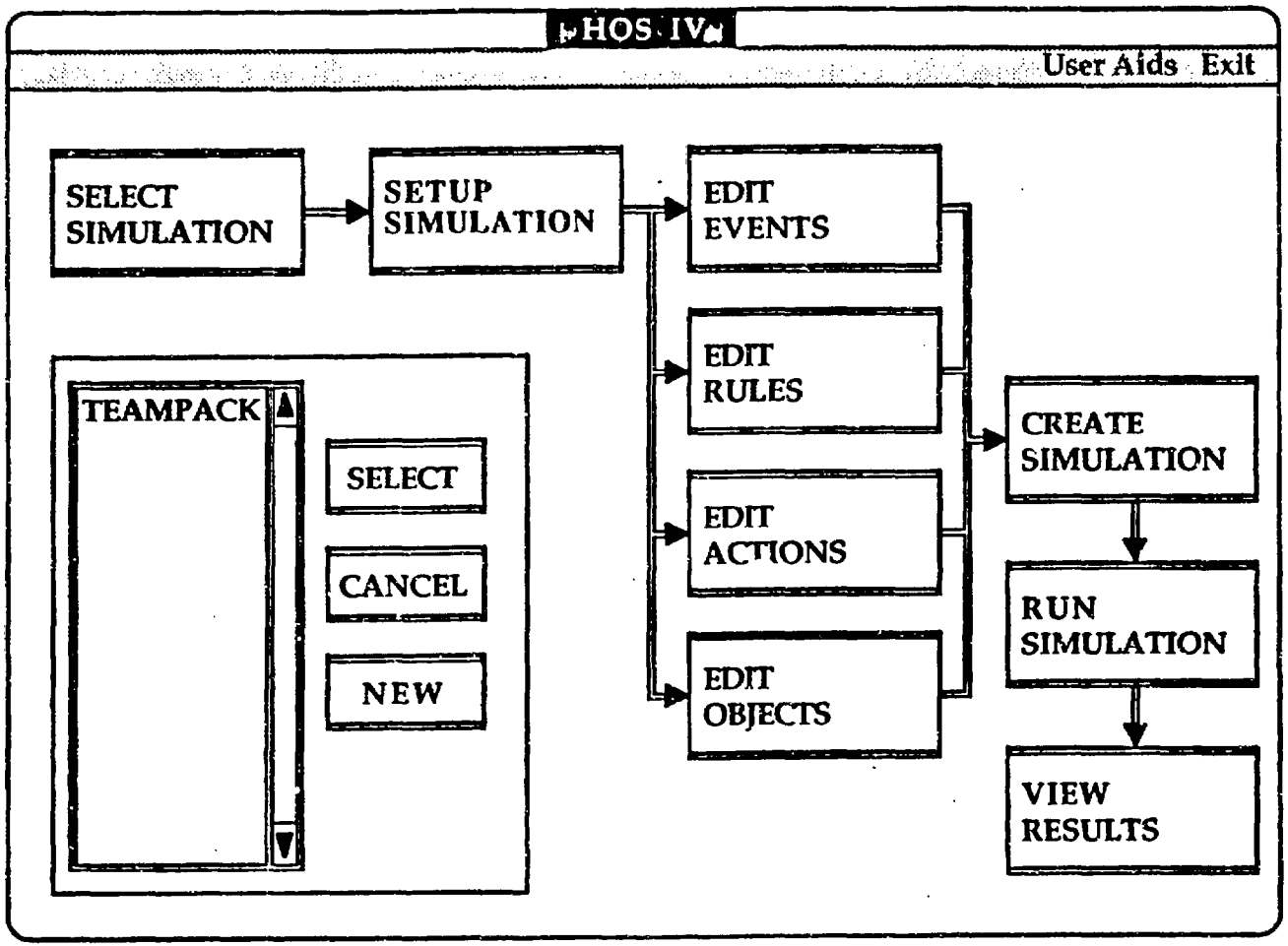


Figure 3-3. Select Simulation Window

The **User Aids** pull-down menu contains commands that allow the user to print the list of simulation names and obtain help messages. They are as follows:

- **Print simulation names** — allows the user to obtain a printout of names of all defined simulations on the line printer.
- **Help** — allows the user to obtain additional information about using SELECT.

SELECT Windows. The main SELECT window is a dialog window for entering object information. The SELECT dialog window, as illustrated in Figure 3-3, contains the following pushbuttons:

- **SELECT** — makes the current simulation the simulation name selected in the list selection box and writes the name to the file d:\HOS-IV\currsim.dat
- **CANCEL** — terminates the SELECT function and returns the user to the main HOS-IV window without changing the current simulation. If no simulation has been selected during the session, a warning message will be displayed.
- **NEW** — defines a new simulation name and generates the new simulation dialog window.

The list selection box contains the list of currently defined simulation names.

When the user selects the **NEW** button from the SELECT dialog window, a new simulation window is displayed as shown in Figure 3-3. The user can select from the following pushbuttons:

- **CANCEL** — cancels the new simulation function.
- **OKAY** — validates the entered simulation name to ensure that it does not contain any illegal characters and has not been previously defined. If the simulation name is valid, the simulation name is added to the list and the simulation directories are created.

The alphabetic dialog window also contains a text entry box for entry of the simulation name.

SELECT Message windows. SELECT generates the following message windows.

- **No simulation selected** — informs the user that a simulation must be selected before HOS-IV can be run. The user options are to continue and select a simulation or exit from HOS-IV.
- **Name already used** — informs the user that the simulation name just entered is not unique and must be re-entered. The user must hit the okay button to return to the Define new simulation window.
- **Invalid character** — informs the user that the simulation name just entered contains an invalid character and must be re-entered. The user must hit the okay button to return to the Define new simulation window.

3.2.2.3 Input/Output

The names of all existing simulations are stored in the file d:\HOS-IV\all_sims.dat. SELECT first determines if this file exists. If it does, the list of simulation names contained in the file are displayed in the list selection box; otherwise it prompts the user to enter the name of a new simulation. The name of the selected simulation is stored in the file d:\HOS-IV\currsim.dat. The contents of these files are shown below:

d:\HOS-IV\all_sims.dat

This file contains a list of eight character simulation names with each name in a separate record.

```
char        name        [8];
```

d:\HOS-IV\currsim.dat

This file contains the eight character simulation name of the current simulation with the name stored in a separate record.

```
char        name        [8];
```

3.2.2.4 Error handling

Message windows are generated for the following conditions:

1. Simulation name contains invalid characters.
2. Simulation name has been previously defined.

3. A simulation must be selected before any HOS processing can occur.

3.2.2.5 Maintenance Procedures

Select simulation is part of the d:\HOS-IV\HOS-IV.exe program. The SELECT source code is contained in the files:

HOS-IV.c
hos_menu.c

In addition, the following files contain global and prototyping information:

HOS-IV.e
HOS-IV.I
hos_menu.e
hos_menu.I

These C files must be compiled using the large model and then linked with the following five libraries in order to produce the executable code:

skyl.llb
ct.llb
te.llb
llibfp.lib
HOS-IV.llb

3.2.3 Setup Simulation

The Setup Simulation module of HOS maintains general information needed to run each simulation. This information includes the minimum time unit, start time, maximum simulation time, start action, and simulation description.

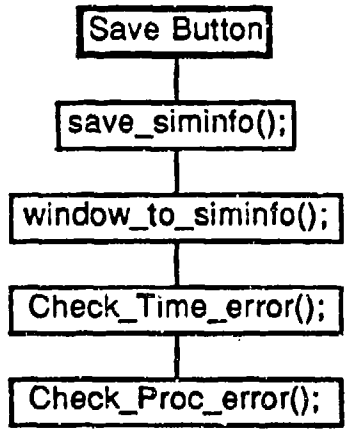
3.2.3.1 Description

The purpose of the Setup module is to define initialization information needed to run each simulation. To execute the Setup Simulation module, the user can depress the SETUP pushbutton from the main HOS-IV module. When the Setup Simulation module is executed, a dialog window is created with entry

fields created for each required piece of information. These fields include the minimum time unit, the simulation start and maximum times, the start action, and the simulation description. The minimum time unit can be one of seven values: days, hours, minutes, seconds, 0.1 seconds, 0.01 seconds, and 0.001 seconds. The user scrolls through a list containing these choices. The maximum simulation time and start simulation time consist of seven fields: days, hours, minutes, seconds, tenths of seconds, hundredths of seconds, and thousandths of seconds. Only those fields greater than or equal to the minimum time unit can be accessed by the user. Invalid fields are blocked out by a blue box covering the field. The user can enter any 80 character string for the simulation description. The start action is any valid defined HOS action.

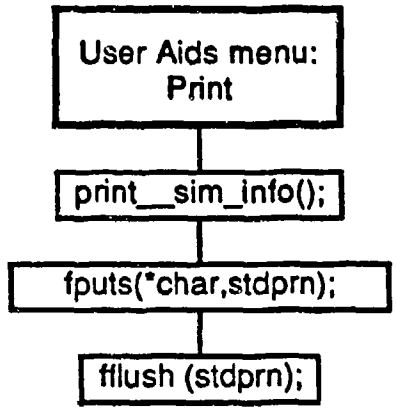
The high-level functional diagrams are organized by user actions and are described below.

Save Button. When the user pushes the save button, the "save" function is executed. The verifying consists of checking for a time error and an action warning. The temporary simulation information record created by verifying process is deleted and the new simulation information is inserted into the global simulation record. This process is illustrated below:



User Aids Menu: Print. When the user selects print from the user aids menu, the "print_sim_info" function is executed. It uses the current simulation information for the printing. This function calls the "fputs" function for every line

that needs to be output such as the times and start action. The process is illustrated below:



It is possible that the user will enter a time that, when converted from a long integer to a string, in terms of the minimum time unit, will have more than six characters. This may cause problems for other modules that expect the time strings to be six characters long. Consequently, a time cannot be greater than 999,999 in terms of the minimum time unit. For example, suppose that the minimum time unit is thousandths of a second. If the user enters a 17 in the minutes field, this will be converted to the string '1020000'.

3.2.3.2 SIMSET Screens

Title Bar. The title bar of SIMSET contains the words 'Setup Simulation' as the function name. SIMSET does not use the current activity or status information areas of the title bar.

SIMSET Menu Bar. SIMSET contains the following menu options on the menu bar as illustrated in Figure 3-4:

- **User Aids** — provides the capability to print the setup information, view the contents of any action file, and obtain help messages, and
- **Exit** — terminates the setup simulation module and returns the user to the HOS-IV screen.

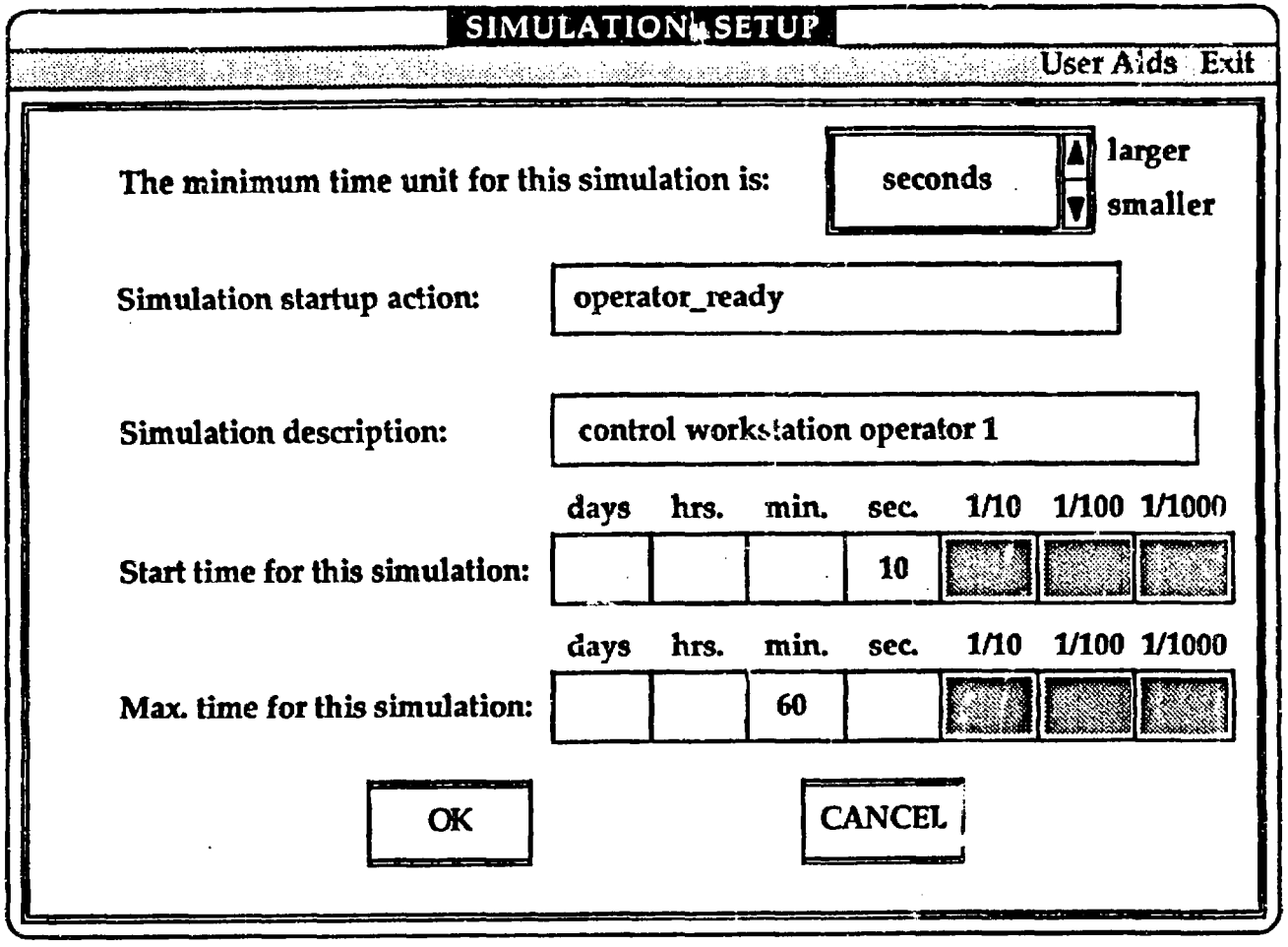


Figure 3-4. Simulation Setup Screen

The **User Aids** pull-down menu contains commands that allow the user to view other files, print current setup simulation information, and obtain help messages. It contains the following commands:

- **View Files** — allows the user to obtain a window that displays currently defined actions.
- **Print** — allows the user to obtain a printout of setup simulation information.
- **Help** — allows the user to obtain additional information about using the setup simulation module.

SIMSET Windows. The main SIMSET window is a dialog window for entering setup simulation information. The SIMSET dialog window, as illustrated in Figure 3-4, contains the following pushbutton:

- **SAVE** — saves the current setup simulation information as displayed on the screen. The following validation is performed prior to the actual saving of the event definition:
  1. The start time is compared to the maximum simulation time. If the start time is greater than the maximum simulation time, an error message window is displayed indicating that the start time is too large.
  2. The start action is evaluated to determine if the action name is valid and has been defined. If it has not, a warning message window is displayed indicating that the action is undefined.

The SIMSET dialog window contains the following text entry boxes:

- **Description** — entry of the simulation description as a maximum of 80 characters.
- **Startup action** — entry of an action name.
- **Maximum Simulation Time** — entry of the maximum simulation time in the form of seven two digit numbers, one for each time unit.
- **Start Simulation Time** — entry of the start simulation time in the form of seven two digit numbers, one for each time unit.

A list selection box showing the minimum time unit is displayed in the upper right corner of the SIMSET dialog window as shown in Figure 3-4.

3.2.3.3 Input/Output

SIMSET uses the following files:

- currsim.dat — contains the current simulation name.
- simname.da1 — contains 'C' code to set the minimum time unit, maximum simulation time, start simulation time, simulation description, and start action. This file is also used as input to the Setup Simulation module to read in the current simulation information.
- simname.da2 — the first record contains an integer (0-7) to indicate the minimum time unit; the second record contains the eighty character simulation description.
- simname.da3 — the first record contains the start time in terms of the minimum time unit; the second and third records contain actual date and time that the simulation started running and stopped running. These last two records are not currently being used.
- simname.da4 — contains the start simulation action name.

3.2.3.4 Error handling

When the user attempts to save the Setup Simulation information, validation checking is performed on the times and start simulation action name. An error is generated if the start time is greater than the maximum simulation time and a descriptive message is displayed in a message window indicating that 'The start time, 00:00:00:00.000, is greater than the maximum simulation time.' Errors must be corrected before a successful save can be accomplished. A warning is generated if the entered action name is undefined. The warning message window is displayed with the message, 'The action _____ has not been defined.'

3.2.3.5 Maintenance Actions

The SIMSET executable is called C:\HOS-IV\simset.exe. The main code for this program is contained in two files:

simset.c

sim_menu.c.

One file is used for global and prototyping information:

simset.i

In addition, the two C files must be compiled using the large model, and then linked with five libraries:

skyl.lib
ct.lib
te.lib
llibfp.lib
HOS-IV.lib

The windows used by EVENTEDIT are stored in the file **simset.cat**.

3.2.4 EVENTEDIT — Edit Events

The EVENTEDIT module of HOS maintains information on all user-defined events. Events are actions that are executed at a user defined time during the simulation to execute some time dependent occurrence. An event consists of an event description, the event action, and the event time at which the event action should be executed.

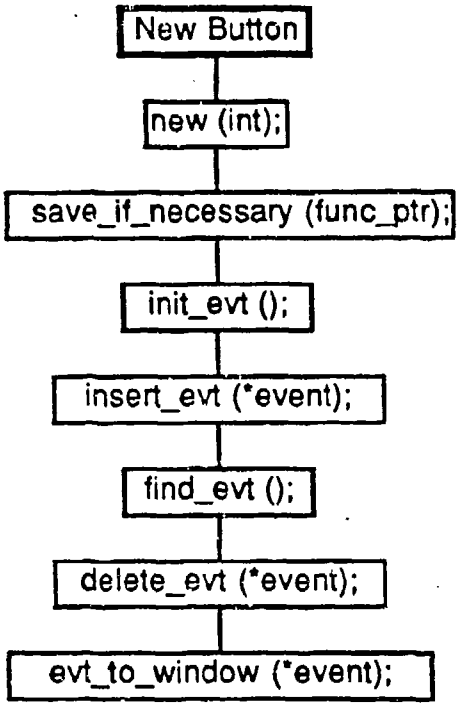
3.2.4.1 Description

EVENTEDIT maintains the list of events for the simulation, permitting the user to view, edit, delete, and create events. The data structure used to maintain the events is a linked list of event records. Each record contains the following elements:

- Event number — Used by the C code generated by HOS as an index for an array of action pointers.
- Event action — A valid defined HOS action name.
- Event time — Consists of a maximum of seven two character strings, one for each time unit.
- Event description — An alphanumeric string containing a maximum of 80 characters.

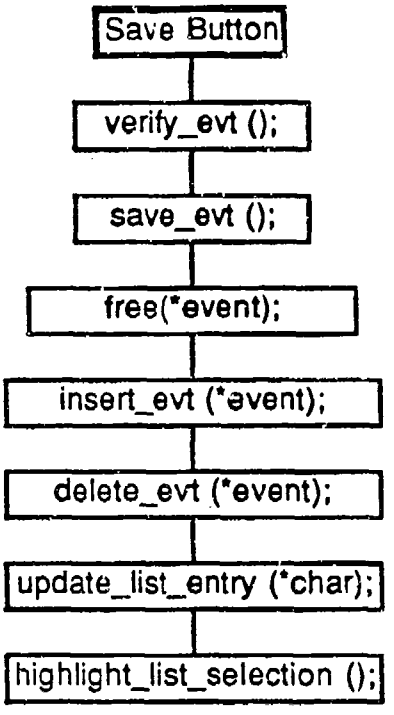
EVENTEDIT is executed from HOS-IV as a result of the user pressing the push button 'Edit Events.' High-level functional diagrams of the EVENTEDIT module are shown below.

New Button. When the user depresses the new push button, the function "new" is executed. It saves the current event, if needed, and then creates and displays a new, blank event. The flow is illustrated below:



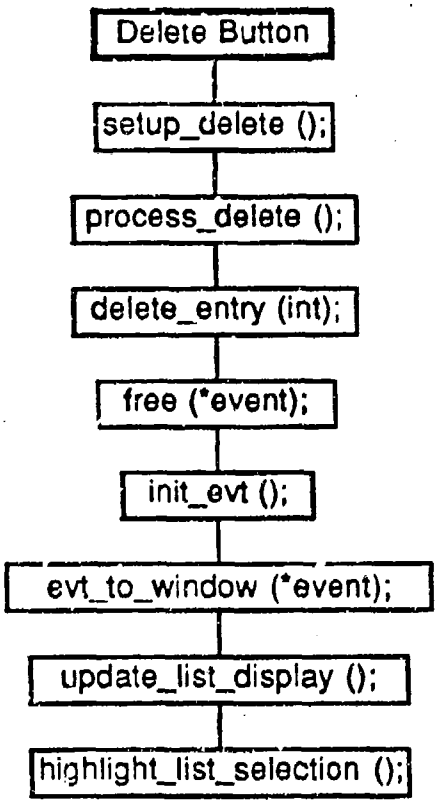
Save Button. When the user pushes the save button, the "save" function is executed. The verify_evt function validates the entered event information and reports any errors. The Save_evt function actually does the linked list maintenance. The temporary event created by verify_evt is deleted and the new event is inserted and then deleted from the linked list of events in order to

establish the pointers. Finally, the event is added to the listbox. This process is illustrated below:



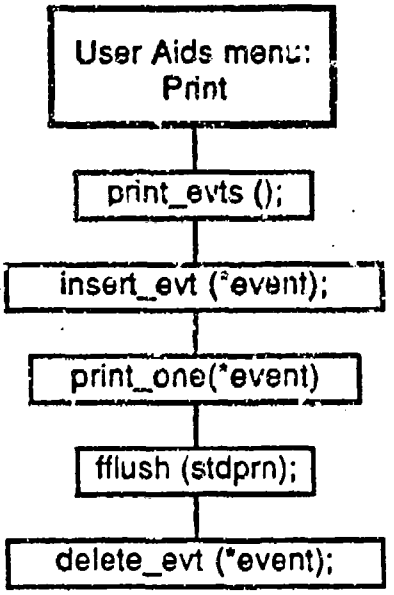
Delete Button. When the user selects an event and then pushes the delete button, the setup_delete function displays the current event and asks the user to verify the delete. Process delete is called if the user verifies. The event is deleted from the list displayed in the list selection box, deleted from the

internal list, and then freed. Finally, the display is returned to normal. This process is illustrated below:



User Aids Menu: Print. When the user selects print from the user aids menu, the "print_evts" function is executed. It inserts the current event into the linked list of events. This function then calls the "print_one" function for every

event in the list. After all of the events have been printed, the event that was inserted is deleted. The process is illustrated below:



3.2.4.2 EVENTEDIT Screens

Title Bar. The title bar of EVENTEDIT contains the words 'Event Editor' as the function name. EVENTEDIT does not use the current activity or status information areas of the title bar.

EVENTEDIT Menu Bar. EVENTEDIT contains the following menu options on the menu bar as illustrated in Figure 3-5:

- **User Aids** — provides the capability to print the list of defined events, view the contents of any action file, and obtain help messages, and
- **Exit** — terminates the Event Editor and returns the user to the HOS-IV screen.

The **User Aids** pull-down menu contains commands that allow the user to view other files, print all currently defined events, and obtain help messages. It contains the following commands:

- **View Files** — allows the user to obtain a window that displays currently defined actions.

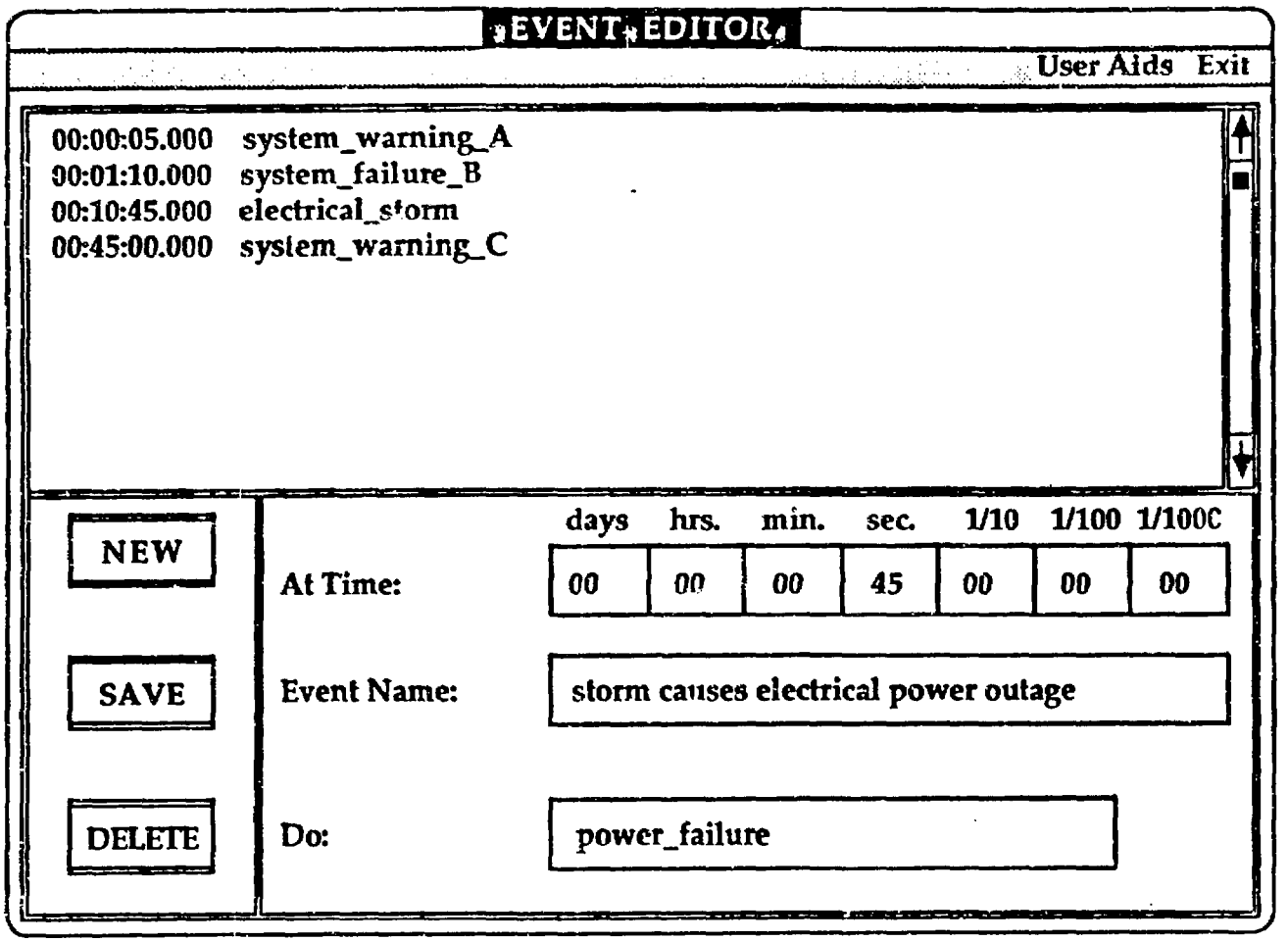


Figure 3-5. Event Editor Screen

- **Print Rules** — allows the user to obtain a printout of all defined events.
- **Help** — allows the user to obtain additional information about using the Event Editor.

EVENTEDIT Windows. The main EVENTEDIT window is a dialog window for entering event information. The event dialog window, as illustrated in Figure 3-5, contains the following pushbuttons:

- **NEW** — clears all input fields and places the text cursor in the event name field.
- **DELETE** — deletes the currently selected event.
- **SAVE** — saves the current event as displayed on the screen. The following validation is performed prior to the actual saving of the event definition:
  1. The event time is compared to the maximum simulation time. If the event time is greater than the maximum simulation time, an error message window is displayed indicating that the event time is too large.
  2. The Do action is evaluated to determine if the action name is valid and has been defined. If it has not, a warning message window is displayed indicating that the action is undefined.

The rule dialog window contains the following text entry boxes:

- **Event Name** — entry of the event name as a maximum of 28 characters.
- **Do** — entry of an action name.
- **At Time** — entry of the time in the form of seven two digit numbers, one for each time unit.

A list selection box showing the event time, description, and event action of currently defined events is displayed in the upper left corner of the event dialog window as shown in Figure 3-5.

3.2.4.3 Input/Output

EVENTEDIT uses the following files:

- **simname.ev1** — used by EVENTEDIT to read in the current defined events. This file is also output when the user quits.

Each record contains four fields separated by a space. A line is in the following format:

- three character event number,
- six character event time,
- thirty-one character event action, and
- eighty character description.

- **simname.ev3** — output by EVENTEDIT. It contains the C code for the array of event action pointers. Each record contains a line in the following format:
  - event_proc[xxx] = zzzzzzzzz;
  - xxx is a number from 0 to number of events - 1. zzzzzzzzz is the event action name.
- **simname.ev2** — output by EVENTEDIT. It contains a record for each event consisting of only the event action name.

3.2.4.4 Error handling

When the user attempts to save an event, error checking is performed on the event time and event action name entered. If the event time is greater than the maximum simulation time, a message window is generated that displays the statement, 'The event time, 00:00:00:00.000 is greater than the maximum simulation time'. Errors must be corrected before a successful save can be accomplished.

If an action name is undefined, a warning message window is displayed with the message, 'The action _____ has not been defined.'

3.2.4.5 Maintenance Actions

The EVENTEDIT executable is called C:\HOS-IV\edit_evt.exe. The main code for this program is contained in two files:

edit_evt.c
evt_menu.c.

Two files are used for global and prototyping information:

edit_evt.I
event.h

In addition, the two C files must be compiled using the large model, and then linked with five libraries:

skyl.llb
ct.llb
te.llb
llibfp.llb
HOS-IV.llb

The windows used by EVENTEDIT are stored in the file **edit_evt.cat.**

3.2.5 RULEEDIT — Edit Rules

The RULEEDIT module of HOS maintains information for all user-defined simulation rules. Rules set up conditions that determine which actions will be executed at a particular simulation time snapshot. A rule is defined by a starting and ending conditional, the name of the action to be invoked if the starting conditional is true, and a unique task number. If the starting conditional statement is true, the named action will be invoked. The action will then continue until the ending conditional is true. Rules can be of three distinct types: operator, hardware, or environment. Each type of rule is grouped separately but identical code is used to maintain the rules.

3.2.5.1 Description

RULEEDIT maintains the list of defined rules for a simulation and permits the user to view, edit, delete, and create rules. There are three sets for rules for each simulation: hardware, operator, and environment. A rule consists of the following elements:

1. Rule Number:
   - Operator — a unique three digit number consisting of a one digit priority assignment from 0 (lowest) to 9 (highest) and a two digit number 00 (lowest) to 99 (highest).
   - Hardware/Environment — a two digit number 00 (lowest) to 99(highest).
2. Rule Name: An alphanumeric name containing a maximum of 28 characters.

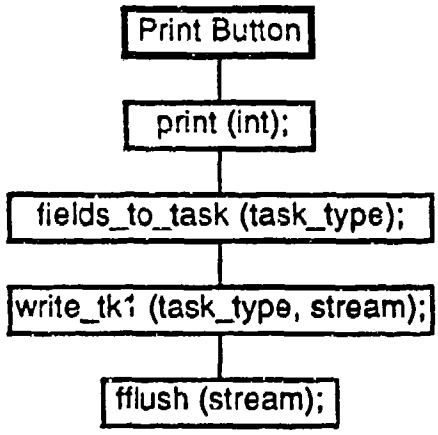
3. If Clause: a starting condition consisting of a Boolean statement utilizing characteristics of objects, constants, and properties.
4. Do Clause: action name.
5. Until Clause: an ending condition consisting of a Boolean statement utilizing characteristics of objects, constants, and properties.

The same code performs operations on all three rule types.

In order to provide the combined files needed by other modules, the Rule Editor combines the hardware, operator and environment files upon exiting.

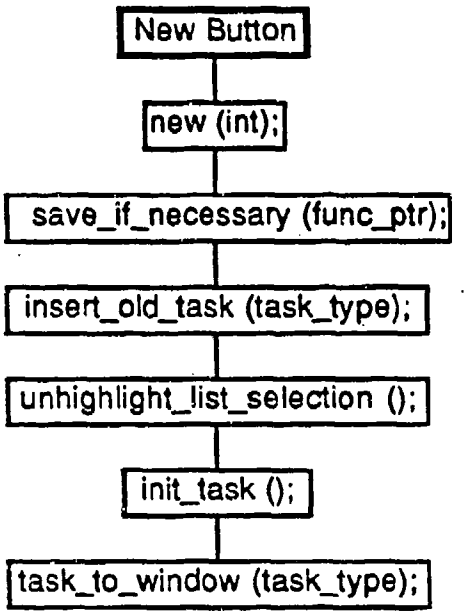
The high-level functional diagrams are organized by user actions and are described below.

Print Button. When the user pushes the print push button, the function "print" is executed. The write_tk1 function writes the current task to the printer stream. The print function is shown below:



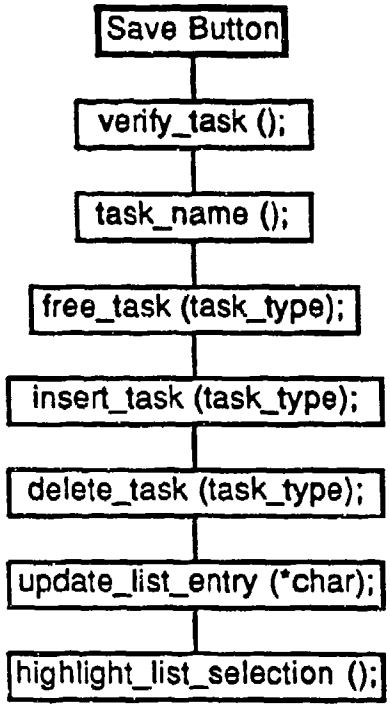
New Button. When the user depresses the new push button, the function "new" is executed. It saves the current task if needed and then creates and

displays a new, blank task. The flow is illustrated below:



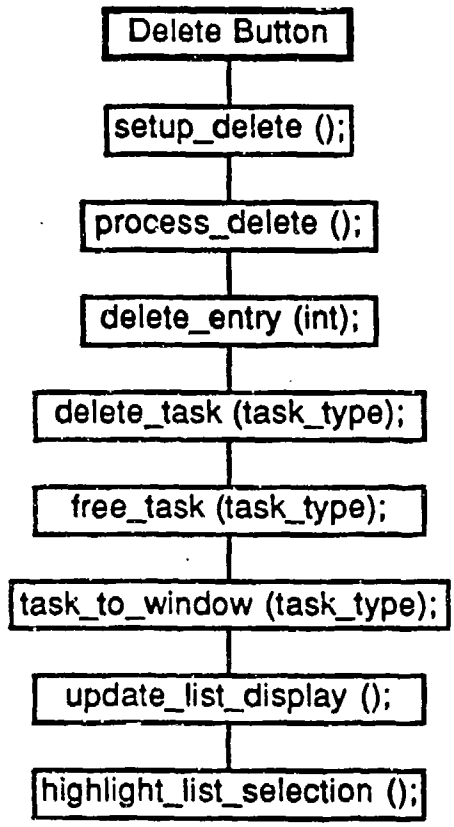
Save Button. When the user pushes the save button, the "save" function is executed. The verify_task function validates the entered task information and reports any errors. Task_name creates the name to display in the list selection box. The temporary task created by verify is deleted and the new task is inserted and then deleted from the task list in order to establish the pointers.

Finally, the task is added to the listbox. This process is illustrated below:



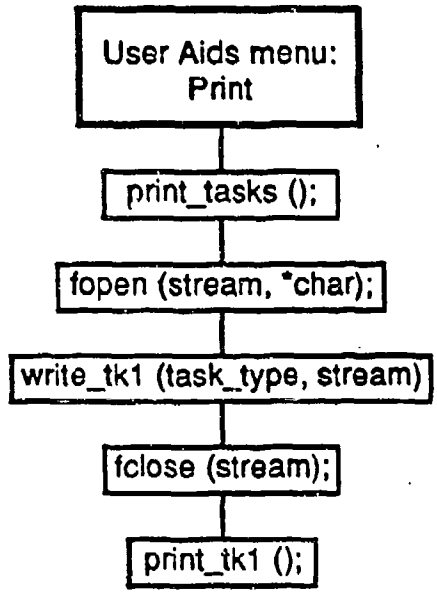
Delete Button. When the user pushes the delete button, the setup_delete function displays the current rule and asks the user to verify the delete. Process delete is called if the user verifies. The rule is deleted from the list displayed in the list selection box, deleted from the internal list, and then freed. Finally, the

display is returned to normal. This process is illustrated below:



User Aids Menu: Print. When the user selects print tasks from the user aids menu, the "print_tasks" function is executed. It opens a file on disk and writes each task to it, formatting it for printing. It then sends the file to the printer.

The process is illustrated below:



3.2.5.2 RULEEDIT Screens

Title Bar. The title bar of RULEEDIT contains the words 'RULE EDITOR' as the function name. RULEEDIT does not use the current activity or status information areas of the title bar.

RULEEDIT Menu Bar. RULEEDIT contains the following menu options on the menu bar as illustrated in Figure 3-6:

- **Rule Types** — defines the rule type as environment, hardware, or operator. When one of the type options is selected, the current rules, if any, are saved and the rules associated with the selected type are loaded.
- **Edit** — provides copy and paste functions. **Copy** copies the current rule into an invisible clipboard buffer. **Paste** copies the clipboard buffer into the rule currently defined on the screen only if the current rule is new.
- **User Aids** — provides the capability to print the list of defined rules, view the contents of any action file, and obtain help messages.
- **Exit** — terminates the Object Editor and returns the user to the HOS-IV screen.

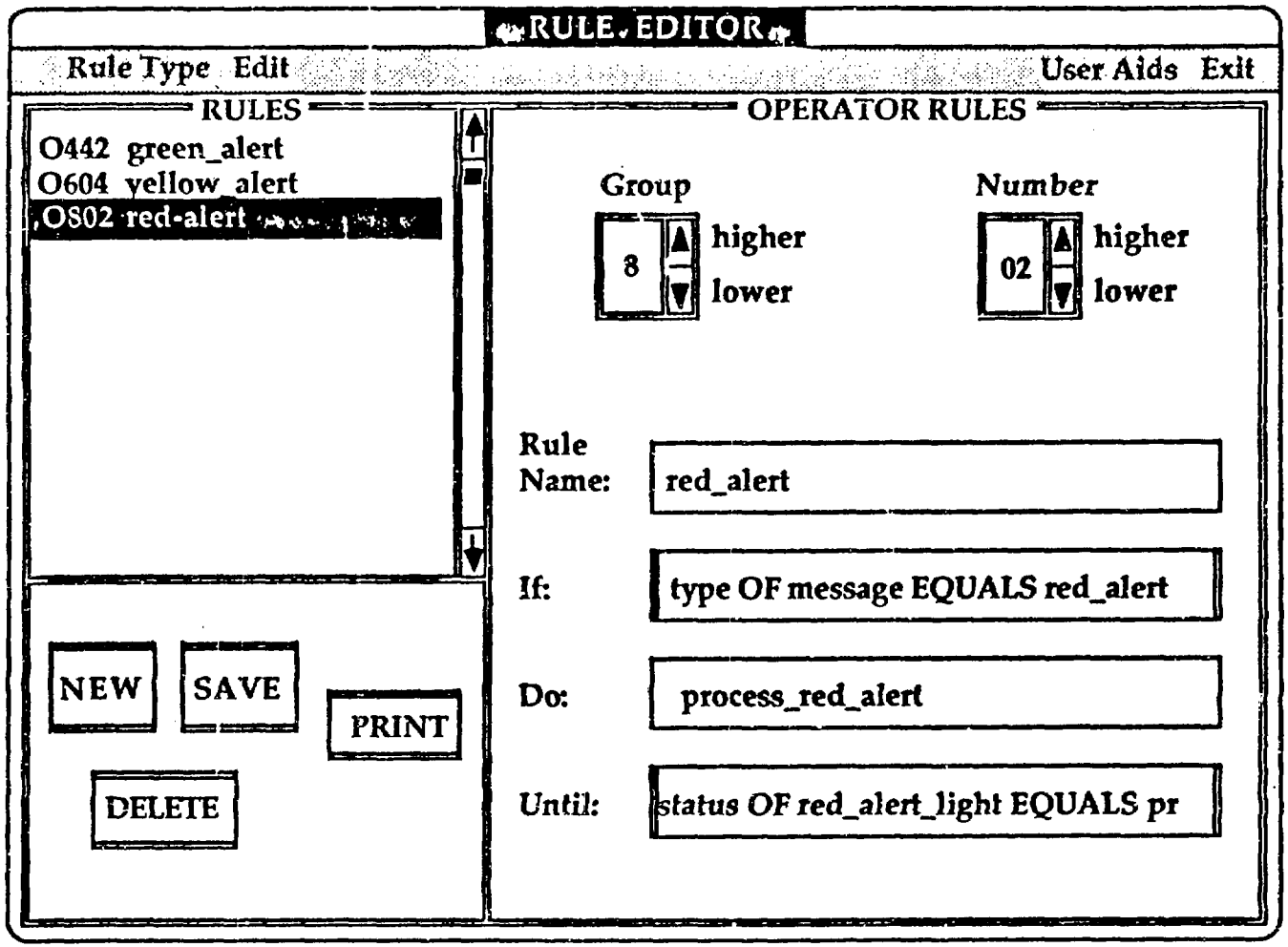


Figure 3-6. Rule Editor Screen

The rule pull-down menu contains commands that allow the user to define the rule type. It contains the following commands:

- **Environment** — defines an environment type rule. All of the data entry fields in the rule dialog window are blanked and the text cursor is placed in the first character in the rule name field. The rule dialog window is modified to remove the Rule Group click box if present.
- **Hardware** — defines a hardware type rule. All of the data entry fields in the rule dialog window are blanked and the text cursor is placed in the first character in the rule name field. The rule dialog window is modified to remove the Rule Group click box if present.
- **Operator** — defines an operator type rule. All of the data entry fields in the rule dialog window are blanked and the text cursor is placed in the first character in the rule name field. The rule dialog window is modified to add the Rule Group click box if not present.

The **User Aids** pull-down menu contains commands that allow the user to view other files, print all currently defined tasks, and obtain help messages. It contains the following commands:

- **View Files** — allows the user to obtain a window that displays currently defined actions.
- **Print rules** — allows the user to obtain a printout of all defined rules. It first formats the rules and then prints it out on the line printer.
- **Help** — allows the user to obtain additional information about using the Rule Editor.

RULEEDIT Windows. The main RULEEDIT window is a dialog window for entering rule information. The rule dialog window, as illustrated in Figure 3-6, contains the following pushbuttons:

- **NEW** — clears all input fields and places the text cursor in the rule name field.
- **DELETE** — deletes the currently selected rule.

- **SAVE** — saves the current rule as displayed on the screen. The following validation is performed prior to the actual saving of the rule definition:
  1. The rule name is a valid name, i.e., it starts with an alphabetic character (a-z), does not contain any illegal characters, and is unique.
  2. The If and Until clauses contain valid Boolean conditions. If the Boolean operator or constant is invalid, an error message window is displayed indicating that the clause cannot be saved. If a characteristic-object pair is invalid, a warning message window is displayed indicating that there is a problem with the Boolean conditions.
  3. The Do action is evaluated to determine if the action name is valid and has been defined. If it has not, a warning message window is displayed indicating that the action is undefined.
- **PRINT** — prints the current rule definition on the printer.

The rule dialog window contains the following text entry boxes:

- **Rule Name** — entry of the rule name as a maximum of 28 characters.
- **If** — entry of the if clause in the form value Boolean operator value where value can be a characteristic of an object or a constant and Boolean operator can be equals, not_equal, less_than, less_or_equal. greater_than, or greater_or_equal.
- **Do** — entry of an action name.
- **Until** — entry of the until clause in the form value Boolean operator value where value can be a characteristic of an object or a constant and Boolean operator can be equals, not_equal, less_than, less_or_equal, greater_than, or greater_or_equal.

The rule dialog window contains a click box for entry of the rule group number 0-9 (operator rules only) and rule number 0-99. A list selection box showing the number and names of currently defined rules is displayed in the upper left corner of the rule dialog window as shown in Figure 3-6.

3.2.5.3 Input/Output

The files produced by RULEEDIT are described below:

- simname.tko — stores the internal representation of the **operator** rules and uses the following record structure for each rule:

```
char        description     [35];
char        if_cond         [200];
char        if_c            [300];
char        procedure       [35];
char        until_cond      [200];
char        until_c         [300];
int         priority;
int         sub_priority;
task_type   *next;
task_type   *last;
```

- simname.tkh — stores the internal representation of the **hardware** rules. It uses the same structure as operator rules.
- simname.tke — stores the internal representation of the **environment** rules. It uses the same structure as operator rules.
- simname.toi — contains C code to initialize the operator tasks.
- simname.thi — contains C code to initialize the hardware tasks.
- simname.tei — contains C code to initialize the environment tasks.
- simname.toc — contains C code to execute the operator tasks.
- simname.thc — contains C code to execute the hardware tasks.
- simname.tec — contains C code to execute the environment tasks.
- simname.to1 — contains the text version of the operator rules. It is formatted for output to the printer including form feeds.
- simname.th1 — contains the text version of the hardware rules. It is formatted for output to the printer including form feeds.

- simname.te1 — contains the text version of the environment rules. It is formatted for output to the printer including form feeds.
- simname.to2 — contains a list of the procedures referenced in the operator rules, each stored in a separate record.
- simname.th2 — contains a list of the procedures referenced in the hardware rules, each stored in a separate record.
- simname.te2 — contains a list of the procedures referenced in the environment rules, each stored in a separate record.
- simname.to5 — contains a list of characteristic object pairs referenced in the operator rules. The characteristic names are separated from the object by a comma and a space and each pair is stored in a separate record.
- simname.th5 — contains a list of characteristic object pairs referenced in the hardware rules. The characteristic names are separated from the object by a comma and a space and each pair is stored in a separate record.
- simname.te5 — contains a list of characteristic object pairs referenced in the environment rules. The characteristic names are separated from the object by a comma and a space and each pair is stored in a separate record.
- simname.to6 — contains a list of all alphabetics referenced in the operator rules; each alphabetic is stored in a separate record.
- simname.th6 — contains a list of all alphabetics referenced in the hardware rules; each alphabetic is stored in a separate record.
- simname.te6 — contains a list of all alphabetics referenced in the environment rules; each alphabetic is stored in a separate record.
- simname.tk2 — contains a list of all procedures referenced in all of the rules. It is a combination of the three files: d:\hosiv\•simname.to2, d:\hosiv\•simname.th2, and d:\hosiv\•simname.te2.
- simname.tk3 — contains the C code to initialize all of the rules. It is a combination of the three files: d:\hosiv\•simname.to3, d:\hosiv\•simname.th3, and d:\hosiv\•simname.te3.
- simname.tk4 — contains the C code to drive all of the rules. It is a combination of the three files: d:\hosiv\•simname.to4, d:\hosiv\•simname.th4, and d:\hosiv\•simname.te4.

- simname.tk5 — contains the list of all characteristic object pairs referenced in the rules. It is a combination of the three files: C:\hosiv*simname.to5, C:\hosiv*simname.th5, and C:\hosiv*simname.te5.
- simname.tk6 — contains the list of all alphabetics referenced in the rules. It is a combination of the three files: d:\hosiv*simname.to6, d:\hosiv*simname.th6, and d:\hosiv*simname.te6.

3.2.5.4 Error handling

When the user completes entering a task and depresses the save pushbutton to save the rule, the entered fields are validated. Error messages indicate errors that must be corrected before the rule can be saved; warning messages are informative messages indicating that something suspect is contained in the rule but it does not have to be corrected before the rule can be saved. The following error messages are generated for If and Until statements:

1. A piece of the statement is missing.
2. Syntax error: (offending token).
3. Undefined alphabetic or syntax error.
4. Syntax error.
5. A task with that number already exists.

Warning messages appear when undefined entities (object, characteristic, or action) are used in the rule definition. The window indicates where the problem is, what type of entity is undefined, and what name the user entered.

3.2.5.5 Maintenance Procedures

The Rule Editor executable is called C:\hosiv\edit_tsk.exe. The main code for this program is contained in the files:

edit_tsk.c
list_tsk.c
io_tsk.c
err_tsk.c
tsk_menu.c

The following files contain global and prototyping information:

edit_tsk.e
list_tsk.e
io_tsk.e
err_tsk.e
tsk_menu.e
edit_tsk.l
list_tsk.l
io_tsk.l
err_tsk.l
tsk_menu.l

The five C files must be compiled using the large model, then linked with the following five libraries to produce the executable code:

skyl.lib
ct.lib
te.lib
llibfp.lib
hosiv.lib

The windows used by RULEEDIT are stored in the following file:

edit_tsk.cat

3.2.6. EDIT_ACT — Action Editor

The Action Editor is used to enter actions. Actions describe what will be done by the operator, system, and environment at a given simulation snapshot if the related rule is true. The steps to accomplish a task which must be performed at a given mission time based on the current environmental and system status are detailed in the action. Actions can include updates to the values of object characteristics, invocation of other actions, and the initiation or suspension of action rules. Actions are the only simulation mechanism which can affect the values of the characteristics of objects.

Actions are defined using a small set of standard verbs (e.g., PERFORM, SET, SUSPEND) known as the HOS Action Language (HAL). A summary of the current set of HOS verbs is shown below:

COMMENT <string> ENDCOMMENT

DEFINITIONS [<def-statement>] ENDDEFINITIONS

END_SIM

FILE [<print-value>] ENDFILE

GET <local> FROM <attribute> OF <object>

IF <boolean> THEN <statement-group> ENDIF {ELSE <statement-group> ENDELSE}

PRINT [<print-value>] ENDPRINT

PUT <send-value> IN <attribute> OF <object>

RETRIEVE <local-object> FROM <set-keyword> <set-name>

SET <local> TO (<formula>)

START <rule number>

STOP <rule number>

SUSPEND <rule number>

USING [<parameter>] DO <proc-name>

WHILE <boolean> THEN <statement-group> ENDWHILE

3.2.6.1 Description

EDIT_ACT is essentially a free format word processor with word wrapping, cut and paste features, and mouse and keypad control of the text

cursor. Once an action is entered, the user can specify that it be translated by HAL for use in the simulation executable. The status of the translation, whether successful or errors were detected, is displayed in a message window. If any errors were detected in the translator, the View File option can be used to create a separate window containing the translator output and accompanying error message.

The high-level functional diagrams organized by pull-down menu options are shown in Figure 3-7.

3.2.6.2 EDIT_ACT Screens

The Action Editor consists of a title bar, a menu bar, a text editing window with a scrollbar, and a number of dialog boxes used for program interaction with the user. The Action Editor screen is illustrated in Figure 3-7.

Action Editor Title Bar. The Action Editor title bar consists of a current activity area on the left which displays the name of the current action, the words 'ACTION EDITOR' in the center, and the current line and column position of the text cursor on the right.

Action Editor Menu Bar. The menu bar for the Action Editor contains the following menu options:

- **File** — file related commands such as saving, opening, etc.
- **Edit** — editing commands: cut, paste, etc.
- **Search** — word and line search commands.
- **User Aids** — provides the capabilities to view help files and action files.
- **Exit** — terminates Action Editor and returns the user to the HOS-IV module.

The **File** pull down menu, illustrated in Figure 3-8, contains:

- **New** — closes the current document after asking if changes should be saved and then creates a new empty action.
- **Open** — closes the current document after asking if changes should be saved and then opens the file selection dialog box.

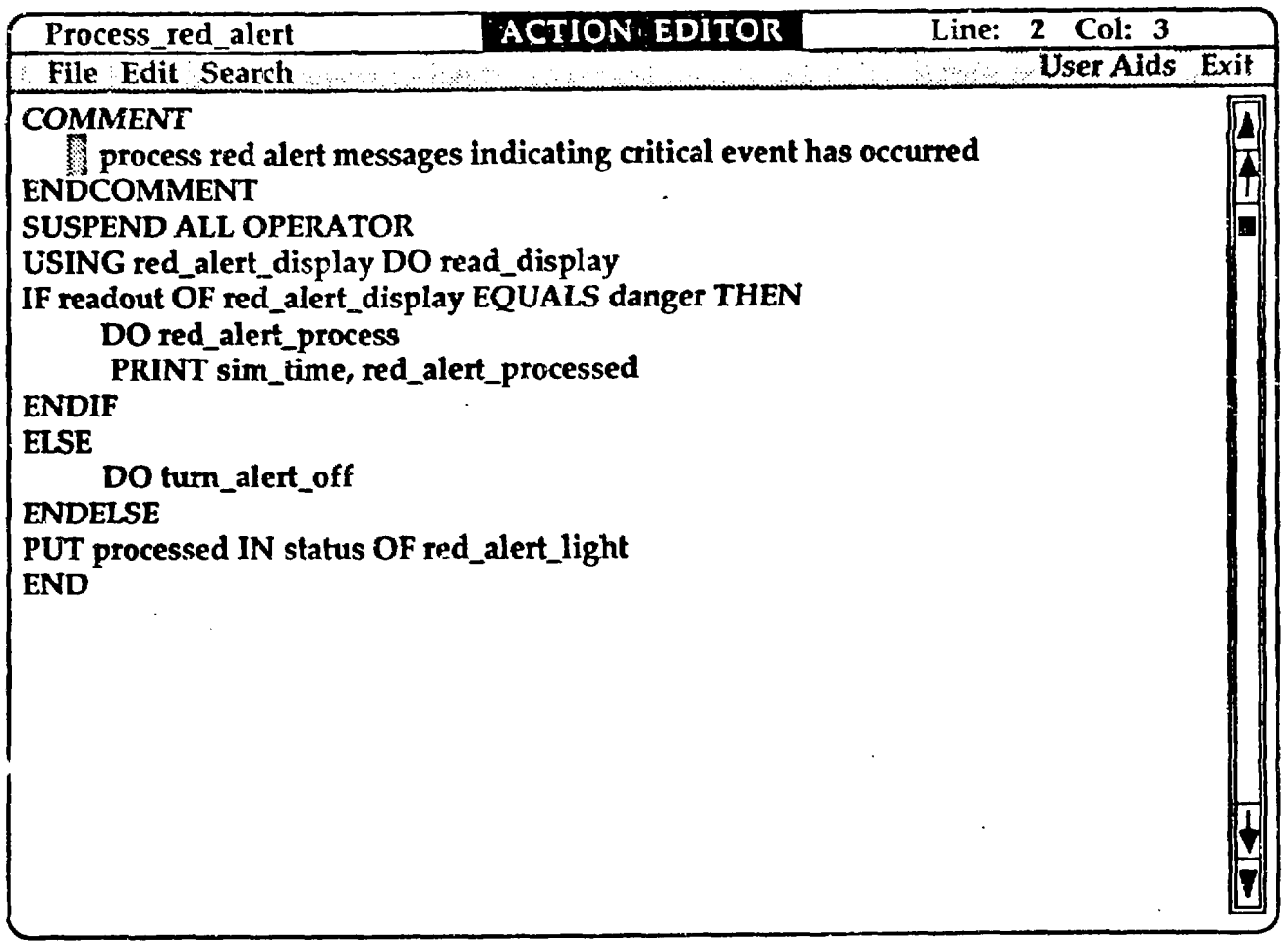


Figure 3-7. Action Editor Screen

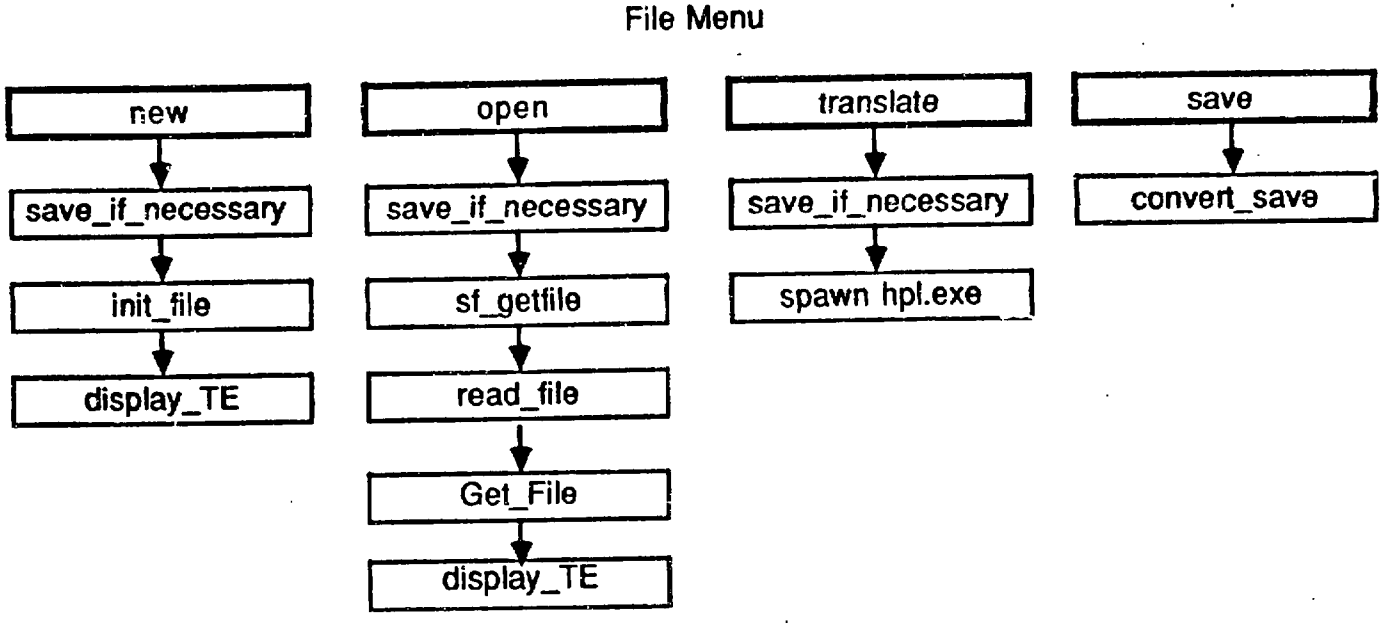


Figure 3-8. Action Editor Functional Diagram

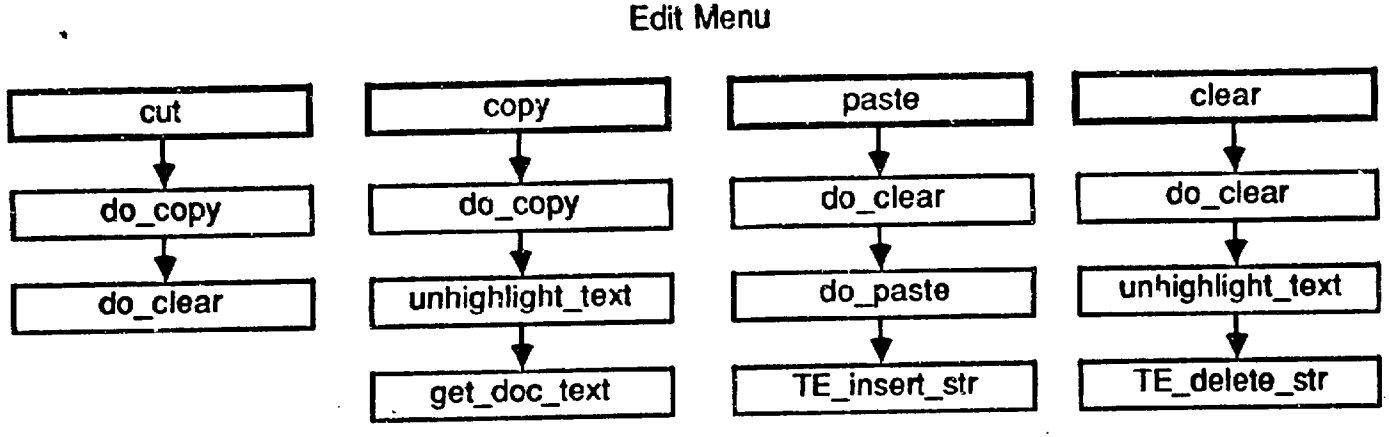


Figure 3-8. Action Editor Functional Diagram (continued)

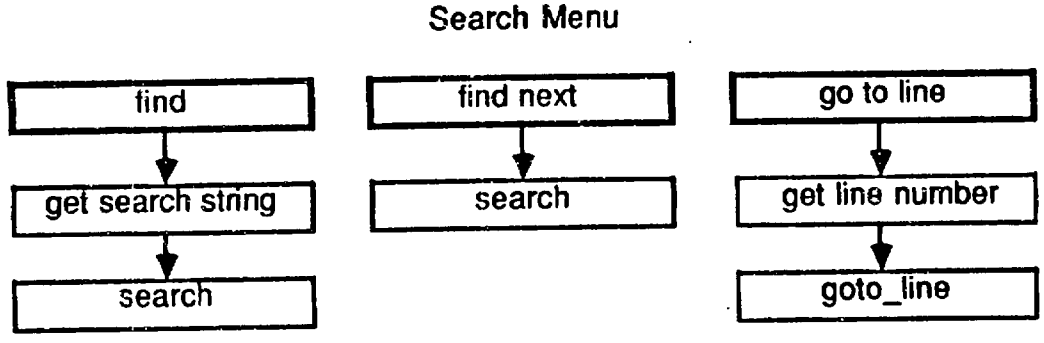


Figure 3-8. Action Editor Functional Diagram (continued)

- **Translate** — asks if changes to the current document should be saved and then spawns the HAL translator.
- **Save** — opens the save dialog box.

The **Edit** pull down menu, illustrated in Figure 3-8, contains:

- **Cut** — removes selected text from the current document and stores it in the clipboard.
- **Copy** — copies selected text from the current document and stores it in the clipboard.
- **Paste** — inserts the contents of the clipboard in the current document at the insertion point.
- **Clear** — deletes the selected text from the current document and throws it away.

The **Search** pull down menu, illustrated in Figure 3-8, contains:

- **Find** — opens the input search string dialog box for the user to define a search string.
- **Find Next** — finds the next occurrence of the string defined in the input search string dialog box.
- **Goto Line** — opens the input line number dialog box for the user to enter the number of the line to goto.

The **User Aids** pull down menu contains commands that allow the user receive help on the current module and view action files:

- **Help** — allows the user to obtain additional information about using ACTION EDITOR.
- **Print** — allows the user to get a formatted hardcopy on the standard printer.
- **View File** — allows the user to view action files created using the Action Editor.

Action Editor Text Editing Window. The text editing window consists of text area and a scrollbar. The text area is 23 rows by 77 columns. The scrollbar is to the right of the text area and has four control buttons and a relative file position indicator. The four controls on the scrollbar are:

- **Scroll Line Down** — displays a page of text starting from the line before the current top line.
- **Scroll Page Down** — displays a page of text starting one page before the current top line.

- **Scroll Page Up** — displays a page of text starting from the current bottom line.
- **Scroll Line Up** — displays a page of text starting from the line after the current top line.

Action Editor Dialog Boxes. Dialog boxes contain text entry fields in addition to pushbuttons, list boxes, and other message window components.

- **File name** — user can edit current action name and choose to save or cancel the save operation.
- **Search string** — user can input string to search for and choose to search or cancel. The search string can contain only alphanumeric characters and underscore (_). It cannot search for control characters such as tabs.
- **Line number** — user can enter integer line number only and choose to move the text cursor to that line or cancel the goto operation.

Action Editor Message Windows. Message windows display information which the user must acknowledge by hitting a pushbutton. Some message windows may be cancelled.

- **String not found** — the search string was not found in the document.
- **Translation successful** — the action was successfully translated.
- **Translation unsuccessful** — the action did not translate.
- **End editing session** — the user may confirm or cancel his Quit selection from the Exit menu.
- **Save changes** — the user may confirm or cancel the saving of the current action.
- **Out of memory** — the document is too large (actually if the file is this big, the compiler won't accept it anyway).
- **File selection** — the user can select an action to open or delete or choose to cancel his selection.
- **Confirm delete** — the user can confirm or cancel his decision to delete an action.
- **Can't delete current file**— the user attempted to delete the currently opened file.
- **Can't delete system file** — the user attempted to delete the TRANSLATOR OUTPUT file or the SYMBOL TABLE file.

Action Editor Information Windows. Information windows do not require any input from the user. They contain informative messages to let the user know that selected commands are being processed.

- **Printing in progress** — the printing of an action is underway.
- **Reading file** — the file is being read.
- **Translating** — the HAL translator is running.

3.2.6.3 Input/Output

Files. EDIT_ACT references the following files:

- **d:\sortproc.p$** — sorted list of action names.
- **d:\htemp.txt** — temporary file created for saving and printing purposes.
- **c:\hosiv\hoshpl\pnnnnnnn.hpl** — actual action files where nnnnnnn is a seven digit sequential number assigned by the EDIT_ACT to uniquely identify the file. When the translated version of the action is store in the pnnnnnnn.c file, the same seven digit identifier is used.
- **c:\hosiv\hoshelp\prc_help.000** — list of help topics specific to the Action Editor.
- **c:\hosiv\hoshelp\prc_help.nnn** — help files (numbered extension starting from 001)

User Actions. EDIT_ACT uses the following function keys:

- **F1** - Begin Mark — sets beginning of text selection
- **F2** - End Mark — sets end of text selection
- **F3** - Cut — see section 3.2.6.1.2 under the **Edit** menu
- **F4** - Copy — see section 3.2.6.1.2 under the **Edit** menu
- **F5** - Paste — see section 3.2.6.1.2 under the **Edit** menu
- **F6** - Clear — see section 3.2.6.1.2 under the **Edit** menu
- **F7 - F10** — Not implemented

3.2.6.4 Error handling

Errors generated by the HAL translator are available to the user in a separate view window. An error flag is passed from HAL to indicate whether or not any errors were detected in the translation process.

TRANSLATOR_OUTPUT contains the translated action and descriptive errors messages.

3.2.6.5 Maintenance Procedures

Action Editor source code (edit_prc.c and prc_menu.c) is compiled using Microsoft C version 4.0 with the large memory model switch (/AL). It requires the HOS-IV, SKYL, CT, and TE libraries in addition to the standard C libraries for linking.

3.2.7. **EDIT_OBJ** — Object Editor

All knowledge about entities to be modeled in a simulation (e.g., displays, controls) are defined as objects. HOS utilizes an object-attribute structure to manage the object data. Each object has an associated list of attributes (e.g., size, location) and each attribute is assigned a value. These attributes describe the important features or characteristics of an object. In order to enhance the user's understanding of this structure, attributes are referred to as characteristics in HOS. Values indicate the state of the characteristic at a particular point in the simulation.

Objects are stored in a library that is accessible to/from all simulations developed on a particular microcomputer. This object library provides a common facility for storing object knowledge and sharing object definitions between simulations. Whenever an object is used [illegible] action, the current object definition is retrieved from the object lib[illegible]nt to note that the object library can be shared by multiple [illegible]J is **not** simulation dependent. This section describes the ED[illegible]odule of HOS which maintains the object and alphabetic library.

3.2.7.1 Description

EDIT_OBJ processes all user actions related to object definitions and their associated characteristics and values. Characteristics types are **whole**, **decimal**, or **alphabetic**. Whole and decimal rep[illegible] numeric values. Alphabetics are text strings and the list of [illegible]abetics are stored in a

separate alphabetics dictionary. EDIT_OBJ maintains the object and set libraries and the alphabetics dictionary. An object definition consists of the following information:

1. **Object Name** — an unique identifier of up to 28 characters. The first character must be an alphabetic (a-z) and the remainder can contain alphabetics (a-z), numbers (0-9), and underscore (_); and
2. **Characteristic List** — list of characteristic names with associated type and initial value. A maximum of 15 characteristics can be defined for an object. The characteristic name can be a maximum of 28 characters including alphabetics (a-z), numbers (0-9), and underscore (_).

The characteristic type is automatically assigned based upon the contents of the initial value. Type whole represents an integer number containing only the digits 0-9 and an optional preceding plus (+) or minus (-) sign. Whole values are saved as C type long that are stored in 4 bytes. The valid range of values for wholes is -2,147,483,648 to 2,147,483,647. Type decimal represents a decimal number that contains only the digits 0-9, a single decimal point (.), and an optional preceding plus (+) or minus (-) sign. Decimal values are saved as C type double that are stored in 8 bytes. The valid range of values for decimals is approximately 1.7E-308 to 1.7E+308. If the value is neither whole nor decimal, then EDIT_OBJ assumes that it is an alphabetic value. Alphabetic values are strings of up to 28 characters that can contain any symbol except space, single quote, or double quote. The strings are entered without any special enclosing characters such as quotes. The alphabetic values are stored in a separate alphabetic dictionary.

Objects can either be defined as:

1. Simple, singular objects such as a single display; or
2. Sets of objects which represent multiple occurrences of identically defined objects such as a list of 10 messages or a set of 500 emitters.

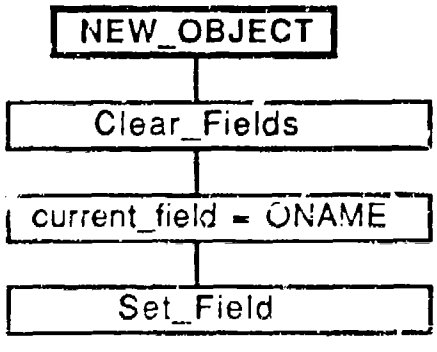
For object sets, the characteristic list must be identical but the value of a characteristic can vary. The names of the members of objects in object sets are constructed by EDIT_OBJ appending a sequential member number, starting with one, to the end of the object set name. For example, the object set name

EMITTER would contain objects named EMITTER001, EMITTER002, EMITTER003, etc. Once a set has been defined, the number of members in the set cannot be changed; i.e., no objects can be added or deleted from the set nor can any characteristic be modified without resaving the entire set. The only item that can be modified by the user is the value of a characteristic. When sets are initially created, all members of the set have the initial value assigned to the set name.

The object library and alphabetic dictionary are accessed by other HOS modules to check for the existence of an object, object/characteristic pair, or an alphabetic value.

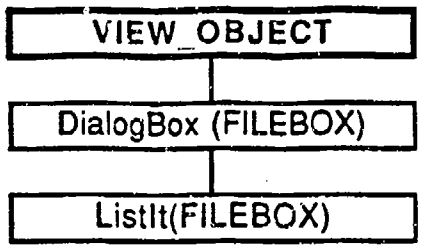
A high-level functional diagram of EDIT_OBJ processes is presented below organized by pushbutton selections.

New Object. When the user depressed the pushbutton labeled "NEW", all text entry boxes on the dialog window are cleared of any previously entered information and the cursor is placed in the object name text entry box. Control is returned to the Edit_driver (keyboard and mouse polling routine) as shown below:



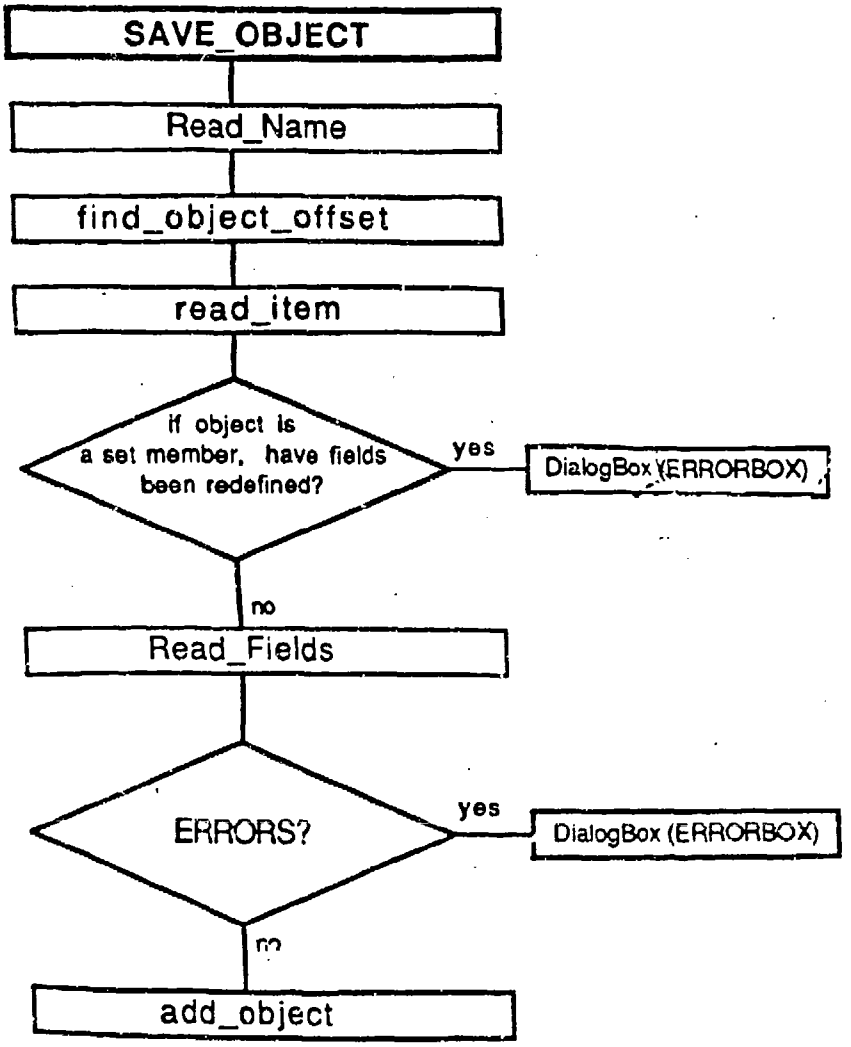
View Object. The view object function displays a list of defined objects in a list selection window and allows the user to select and open or delete an object. If the selected object is a set member, the user is not allowed to delete it or modify the characteristic names or types. For set objects, the user can only change the default values of the characteristics. When view object is

completed, control is then returned to the Edit_driver. View object is illustrated below:



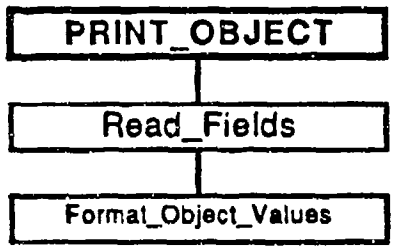
Save Object. The save function performs error checking on the values entered by the user and is illustrated below. Any detected errors are displayed in red and a message window is displayed to indicate the type of error. If the item is a set member, a match is done against the set definition. If it matches, the object will be saved with its new set of values. If the definition has been modified, it will display an error message telling the user that the entire set must be resaved in order to change the set definition. The Read_Field routine assigns one of three types (WHOLE, DECIMAL, or ALPHABETIC) to each characteristic based upon the entered initial value.

If the object is new, the ramdrive directories are recreated and control is returned to the Edit_Driver.

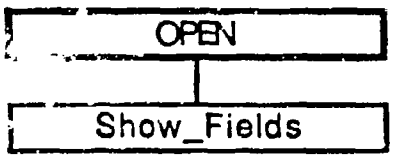


Print Object. The Print_Object function is shown below and reads the values in the edit fields and performs error checking.

If no errors are detected, the object definition will be formatted and sent to the printer. Upon completion of the printing, control is returned to the Edit_Driver.

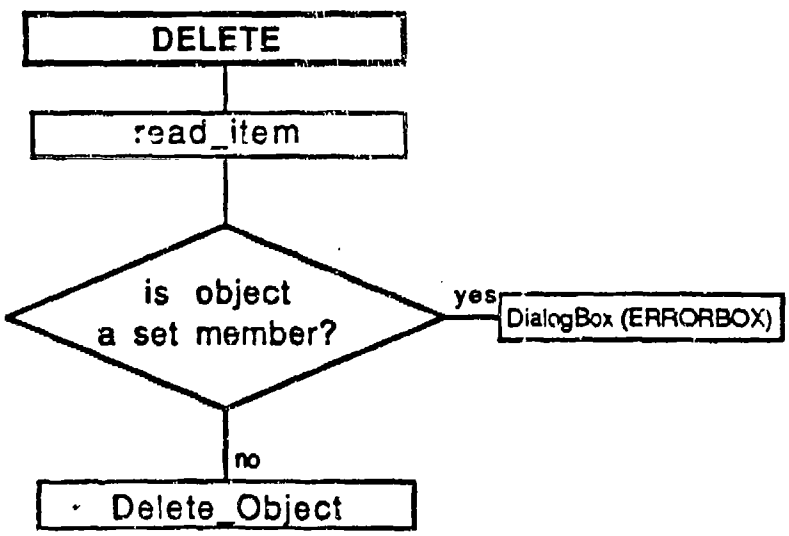


Open Object. The OPEN function searches for the selected object in the directory, reads the object definition, and formats and displays the information in the appropriate field in the dialog window. It is illustrated below:

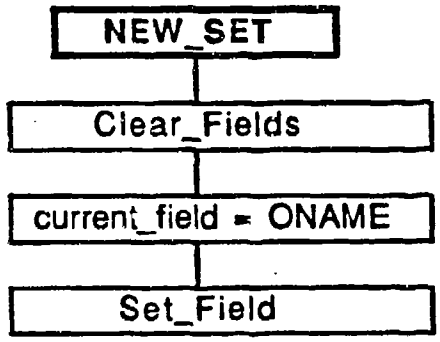


Delete Object. The DELETE option reads the object selected by the user for deletion and displays an error if the object is a set member. Otherwise, the object is marked for deletion.

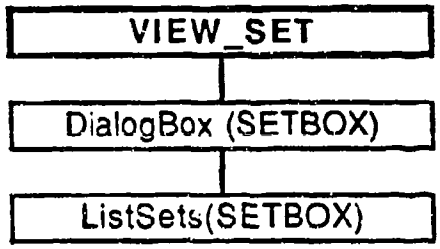
A deletion from the object library is done when the object editing function is completed. DELETE is illustrated below:



New_Set. The New_Set function clears edit fields of previous information, sets the cursor in the object name text entry box, and returns control to the Edit_Driver. It is illustrated below:

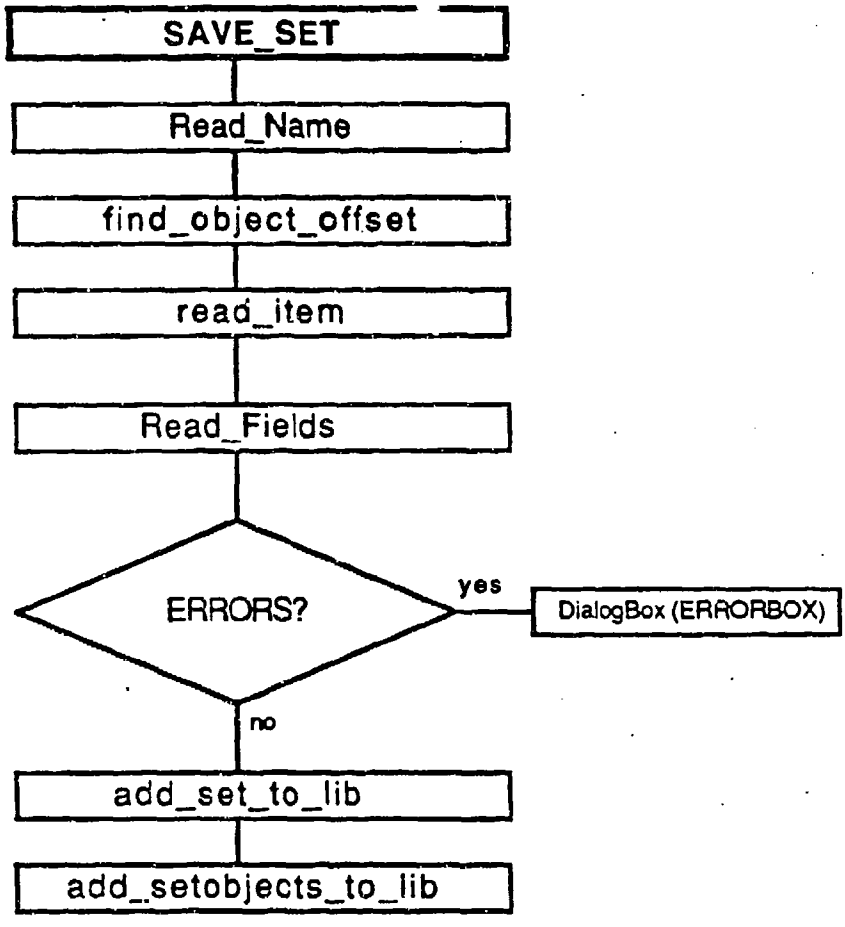


View A Set. The view a set function displays a list of defined sets in a list selection window and allows the user to select a set for opening or deletion. Control is then returned to the Edit_Driver. It is illustrated below:



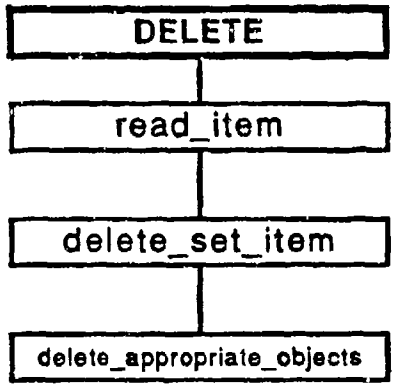
Save A Set. The save function performs error checking on the values entered by the user. If the object values are deemed valid, an entry is made into the set library. Copies of the objects (setname appended with a number from '001' to '999') are added to the object library.

The appropriate ramdrive directories are updated and control is returned to the Edit_Driver. Save_Set is illustrated below:

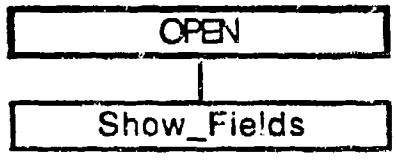


Delete. The DELETE option reads the set name selected by the user for deletion and marks it for deletion.

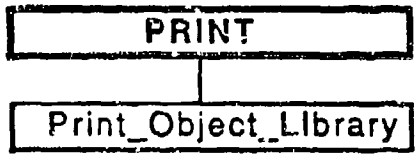
The actual deletion of the set information from the object and set libraries is performed upon termination of the object editing function. DELETE is illustrated below:



Open. The OPEN function searches for the selected set in the directory, reads in the first object of the set, and then formats and displays the fields in the dialog window. Control is then returned to the Edit_Driver. OPEN is illustrated below:

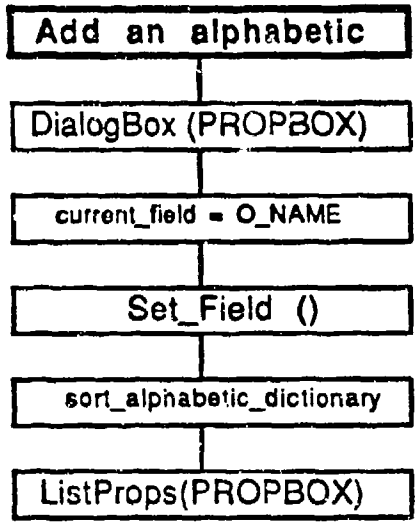


Print. The print object library function formats and prints the entire library to device stdprn. When the printing is completed, control is returned to the Edit_Driver. PRINT is illustrated below:



Add an Alphabetic. The 'Add alphabetic to dictionary' function generates a dialog window with a scrollable list viewing window containing the names of the currently defined alphabetics in sorted order. If insufficient memory is available to sort the alphabetic names, the alphabetics are displayed in random order. It provides a text entry window for entering a new alphabetic. The user is allowed to CANCEL the function and return to Edit_Driver or SAVE

which adds the alphabetic to the end of the alphabetic dictionary. Add_an_alphabetic is illustrated below:



3.2.7.2 EDIT_OBJ Screens

Title Bar. The title bar of EDIT_OBJ contains the words 'OBJECT EDITOR' as the function name. EDIT_OBJ does not use the current activity or status information areas of the title bar.

EDIT_OBJ Menu Bar. EDIT_OBJ contains the following menu options on the menu bar as illustrated in Figure 3-9:

- **Sets** -- defines and manipulates object sets,
- **User Aids** — provides the capability to add a value to the alphabetic dictionary, print the object library, and obtain help messages, and
- **Exit** — terminates the Object Editor and returns the user to the HOS-IV screen.

The **set** pull-down menu contains commands that allow the user to create, save, and view object sets. They are as follows:

- **New set** — defines a new object set. All of the data entry fields in the object dialog window are blanked and the text cursor is placed in the first character in the object name field.
- **Save a set** — saves an object set and creates n objects in the set where n is the number of items in the set specified in the number field. An informative message window is displayed

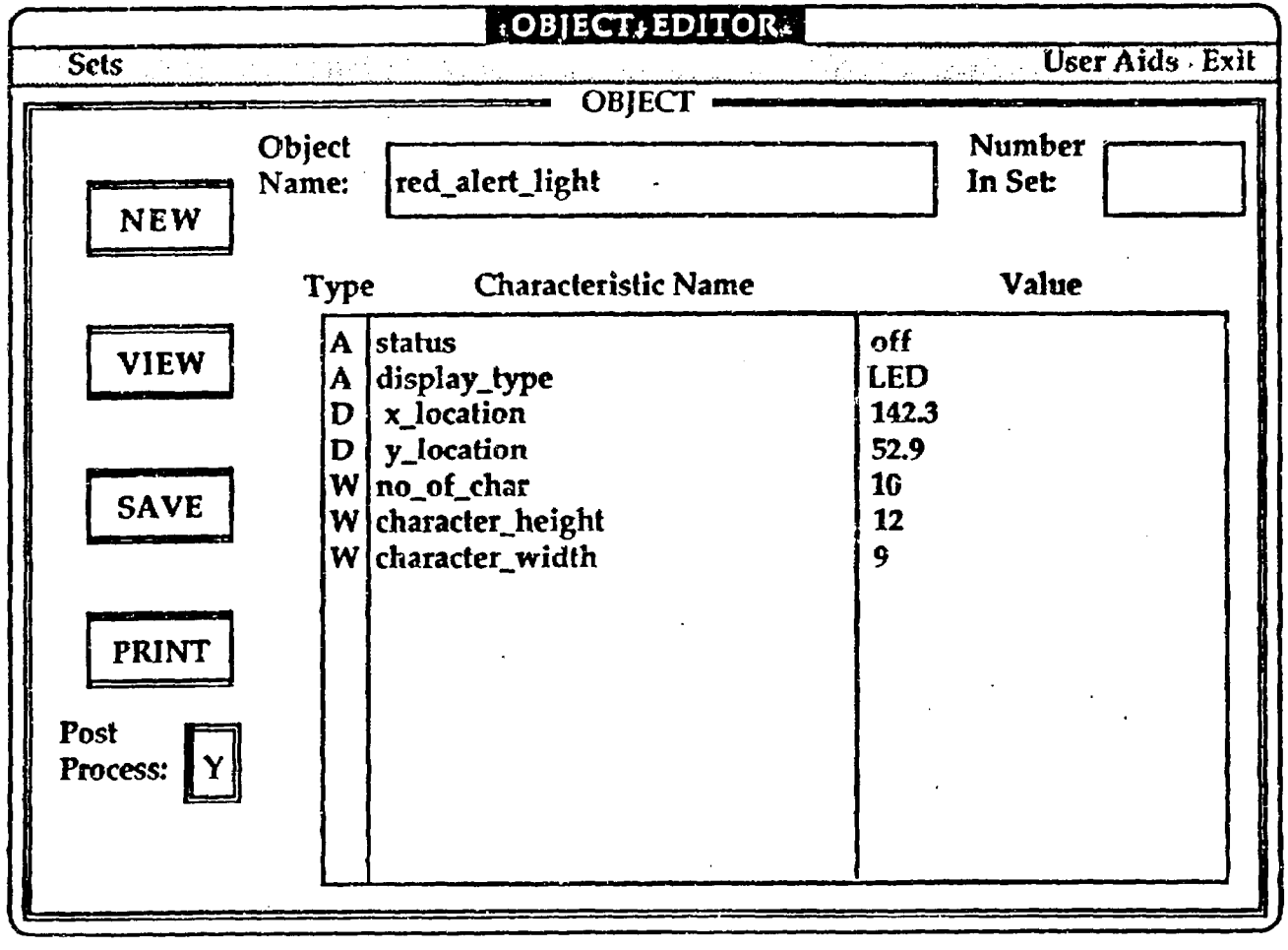


Figure 3-9. Object Editor Screen

showing the object number being created and the number in the set.

- **View a set**— creates a list box containing the name of all currently defined sets and permits the user to select a set by pointing to the desired set name. The user has the option to either (1) delete all members in the set and the object set itself by depressing the **Delete** pushbutton, (2) open the set by depressing the **Open** pushbutton, or (3) **Cancel** the view operation. For the open set option, the contents of the first member of the set (i.e., object number 1 in the set) is displayed in the object dialog window.

The **User Aids** pull-down menu contains commands that allow the user to create, save, and view object sets. They are:

- **Add an alphabetic** — allows the user to add an alphabetic to the alphabetic dictionary. New alphabetics are always added at the end of the dictionary because the alphabetics are implemented by using their position in the dictionary as the value to be placed in the object. However, the alphabetics are presented to the user in alphabetic order.
- **Print the object library** — allows the user to obtain a printout of all objects in the object library. It first formats the object library, and then prints it out on the line printer.
- **Help** — allows the user to obtain additional information about using the Object Editor.

EDIT_OBJ Windows. The main EDIT_OBJ window is a dialog window for entering object information. The object dialog window, as illustrated in Figure 3-9, contains the following pushbuttons:

- **NEW** — clears all input fields and places the text cursor in the object name field.
- **VIEW** — displays a list selection box containing the names of the currently defined objects in the object library (in alphabetic order), including members of set objects. The user can then use the point and click selection method to select an object name. The currently selected object will be highlighted in black. The following pushbuttons are functional in the view window:
  - **OPEN** — closes the view window and displays the current definition of the selected object in the object dialog window.
  - **DELETE** — deletes the object currently selected object. If the object is a member of a set, a message window is

created to inform the user that members of sets cannot be deleted.

- **SAVE** — saves the current object as displayed on the screen. The following validation is performed prior to the actual saving of the object definition:

  1. The object and characteristic names are valid names, i.e., they start with an alphabetic character (a-z) and do not contain illegal characters.
  2. The value of a characteristic must be one of the following:
  - Numeric: either real or integer (with values which can be contained in a long or double precision); or
  - Alphabetic: the value must be in the alphabetic dictionary.
  3. The characteristic type is determined based upon the entered value as described in Section 3.2.7.1. The first letter of the type (W=whole, D=decimal; A=alphabetic) is displayed in the type field of the appropriate characteristic. The default type is WHOLE.
  4. An initial value must be entered for every characteristic name.
  5. For every entered value, a characteristic name was supplied.
  6. If both the characteristic name and value fields are blank, that row of information is ignored.
- **PRINT** — prints the current object definition on the printer.

The object dialog window contains the following text entry boxes:

- **Object Name** — entry of the object name as a maximum of 28 characters.
- **Number In Set** — entry of the number of members in a set. Used only when the user selects the **Save a Set** option from the **Sets** menu bar. Valid entries are a number between 2 and 999.
- **Characteristic Name/Value** — entry of a maximum of 14 pairs of characteristic names and values. The type column is automatically filled in by EDIT_OBJ during the SAVE operation.

When the user selects the **Add an alphabetic** option from the **User Aids** menu bar, an alphabetic dialog window is displayed as shown in Figure 3-10. The alphabetic dialog window contains a list viewing box that displays the list of the currently defined alphabetic in alphabetic order.

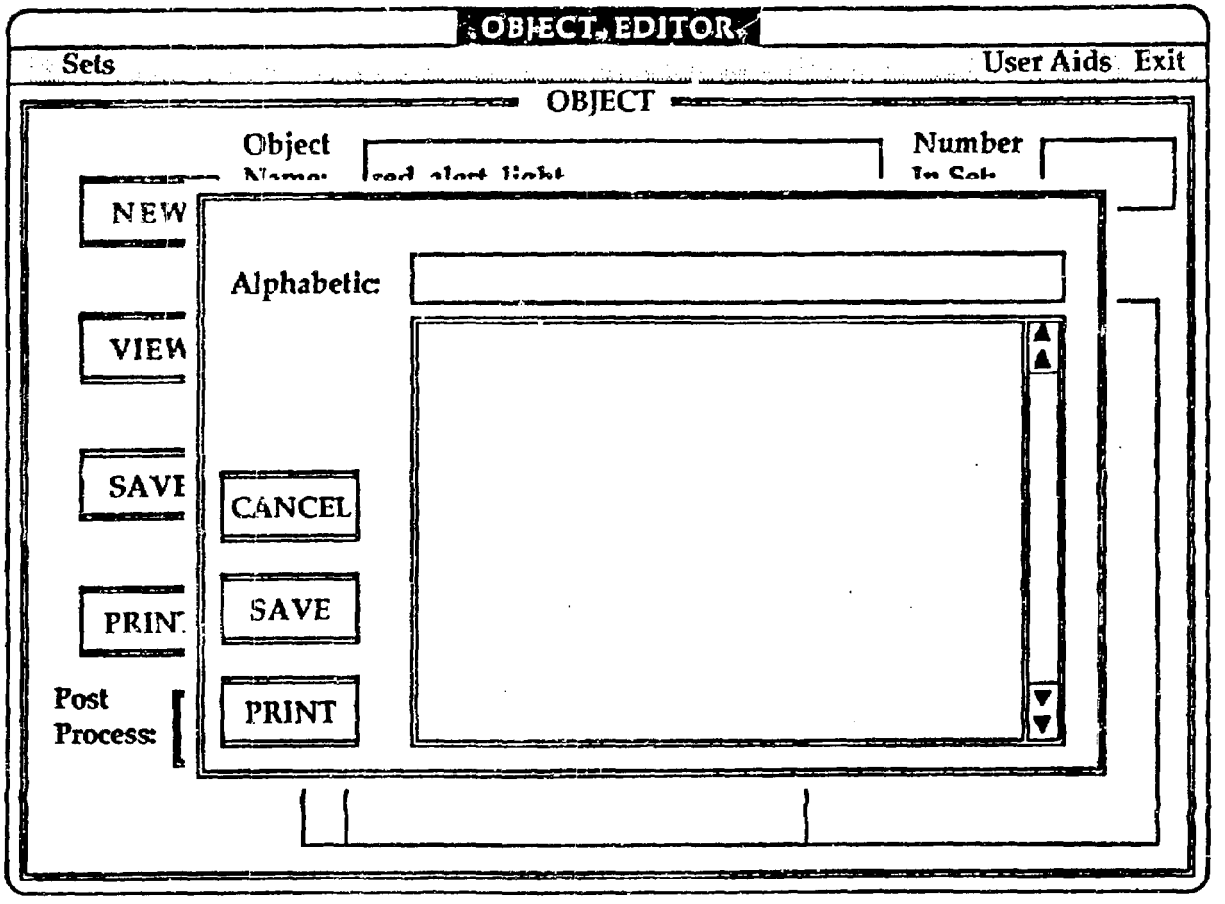


Figure 3-10. Alphabetic Viewing Window

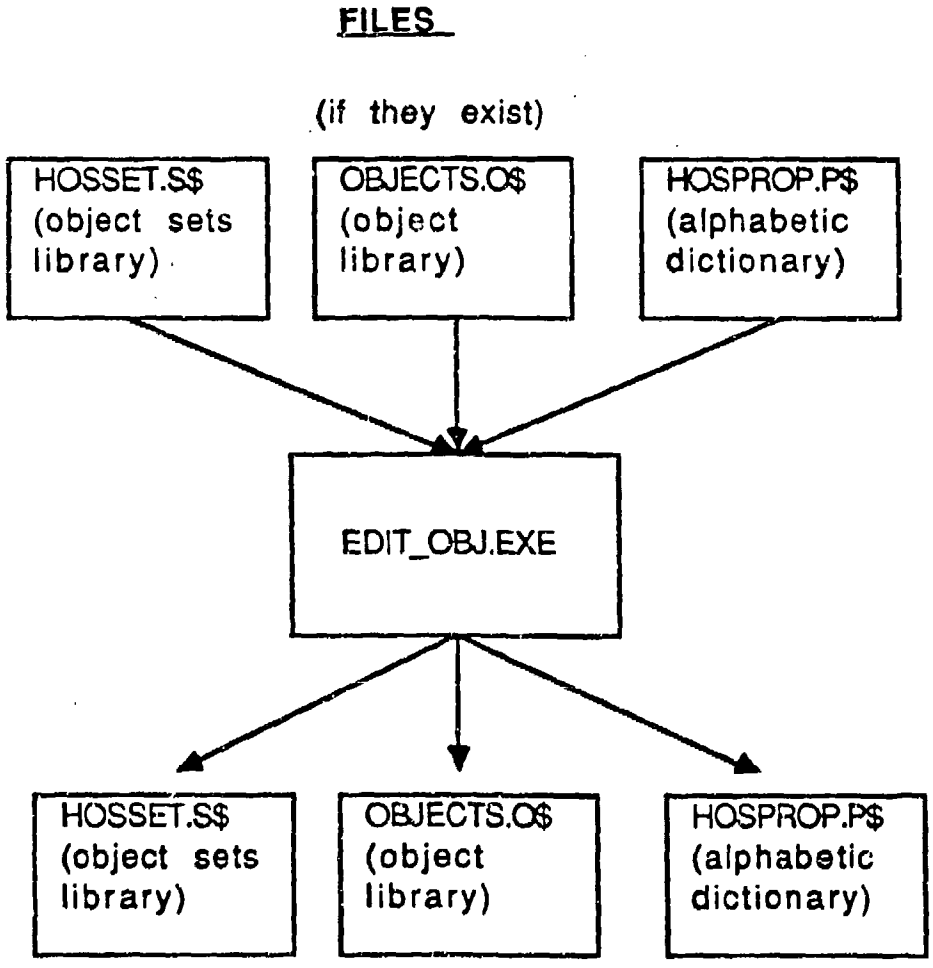
The user can select from the following pushbuttons:

- **CANCEL** — cancels the add an alphabetic function.
- **SAVE** — adds the entered alphabetic name to the alphabetic dictionary.

The alphabetic dialog window also contains a text entry box for entry of the alphabetic name.

3.2.7.3 Input/Output

The files produced by EDIT_OBJ include the following and are illustrated below:



- Objects.o$ — Object library

- Hosprop.p$ — Alphabetic dictionary
- HOSSET.S$ — Set information file containing number of members and Snnnnnn.set (n is between 0 and 9999999) which will be created and used at simulation execution time to contain set information.

Each defined object requires 668 bytes in the objects.o$ file.

3.2.7.4 Error handling

OBJECT errors include the following :

- Failure to open a necessary file. This returns to the calling procedure, but does not cause program failure.
- File read/write failure. Closes all files and returns but does not cause program failure.
- File seek failure. Since random file access is used wherever possible, seek faults can occur. This error closes all files and returns to the calling program, without doing a read, but does not cause program failure.
- All user errors are displayed to the user in a message or information window.
- Memory allocation errors which are recoverable.

3.2.7.5 Maintenance Procedures

OBJECT source code is compiled using MSC Microsoft C version 4.0 with the large memory model switch (/AL). It requires the HOS-IV library and SKYL libraries in addition to the standard C libraries for compilation.

3.2.8. **HAL** — HOS Action Translator

HAL, the HOS Action translator, translates actions into C code. It is invoked by the Action Editor, ACTEDIT, automatically when the user selects the **Translate** option on the **File** pull-down menu as described in Section 3.2.6.

3.2.8.1 Description

The HAL translator is a one pass translator using a forward-chaining translation scheme. The user ACTION file (Pnnnnnn.HPL) which is currently open within the Action Editor is input to HAL by EDIT_ACT. nnnnnnn

represents a unique seven-digit number assigned by the Action Editor to translate the 28 character action name into a valid DOS file name. The translator reads one token at a time. The token is read in as a string which is isolated by valid delimiters such as the comma or a space. Each token is analyzed to determine its type -- for example, it determines if the current token is a HAL verb keyword, an object name, a characteristic name, or a local variable. Depending on the type of token, the translator will analyze the following token and determine if a statement is syntactically correct. A high-level functional diagram of HAL is shown in Figure 3-11.

3.2.8.2 HAL Screens

HAL runs solely as a batch process with all input and output controlled by the Action Editor.

3.2.8.3 Input/Output

The HAL translator produces four files:

- The SYMBOL TABLE file which lists the name and type of the variables used by the action currently being translated. This file is accessible to the user from the VIEW FILE option of the User Aids pull-down menu and is named SYMBOL TABLE.
- The TRANSLATOR OUTPUT file which lists the HAL code and the results of the HAL translation including appropriate syntax error messages. This file is accessible to the user from the VIEW FILE option of the User Aids pull-down menu and is named TRANSLATOR OUTPUT.
- The C statement file which lists the translated HAL code (simname\pnnnnnnn.c) where nnnnnnn is the same seven-digit number assigned to the action name.
- The include file which contains local variable data definitions.

These files can be accessed through (1) the Action Editor by using the VIEW FILE option on the User Aids pull-down menu, (2) by opening the file directly through the file menu in the Action Editor, or (3) by typing it using DOS commands. The files are located in the \HOSIV\HOSHPL subdirectory.

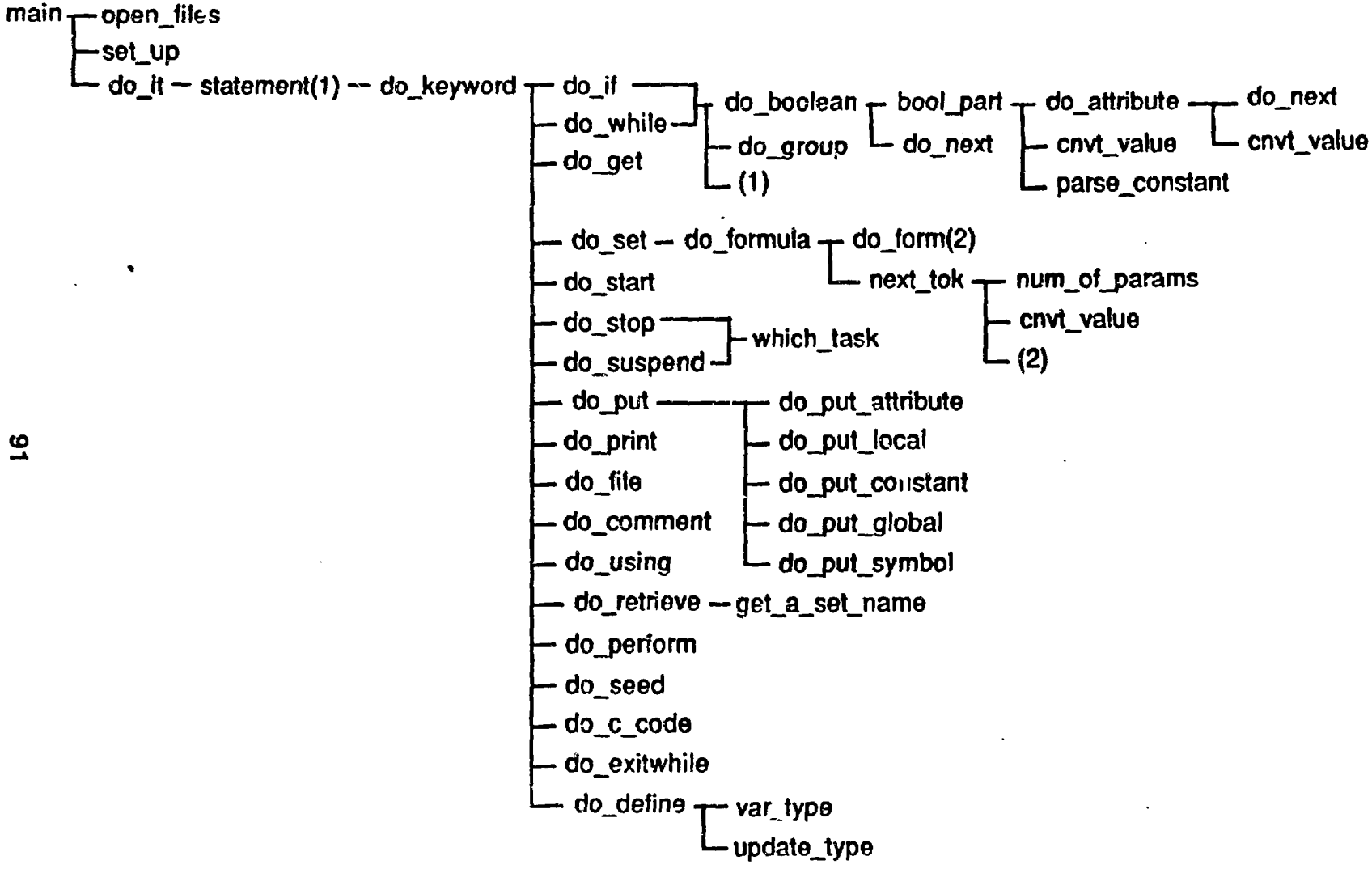


Figure 3-11. HPL Translator Flowchart

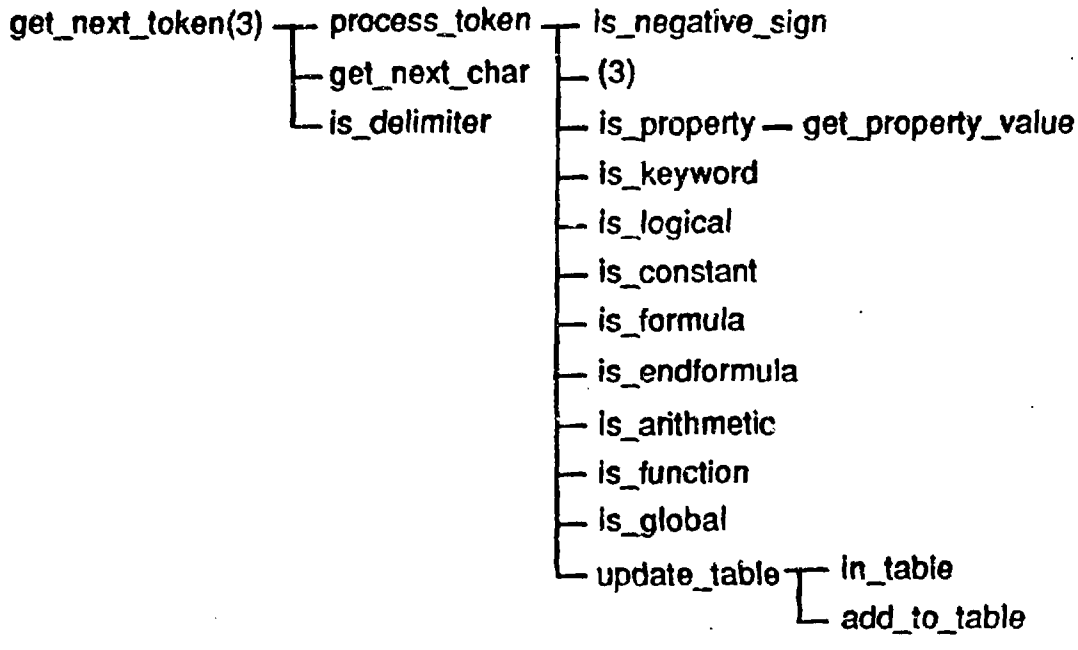


Figure 3-11. HPL Translator Flowchart (continued)

3.2.8.4 Error handling

The HAL error messages are described in Appendix B.

3.2.8.5 Maintenance Procedures

HAL source code is compiled using MSC Microsoft C version 4.0 with the large memory model switch (/AL). It requires the HOS-IV library and SKYL libraries in addition to the standard C libraries for compilation.

3.2.9 CREATE — Create Simulation

3.2.9.1 Description

The CREATE module constructs the HOS simulation based upon the entered events, rules, objects, and actions. It processed all the individual definitions to ensure that each referenced item has been defined. These checks include:

- Ensuring that all actions referenced in events have been defined.
- Ensuring that all actions, alphabetics, and object-characteristic pairs referenced in rules have been defined.
- Ensuring that all object-characteristic pairs, alphabetics, actions, and rules referenced in actions have been defined.

If all cross-references have been validated, then the C code generated for each referenced action is compiled and linked with the HOS simulation C code and produces the executable simulation file. A high-level functional diagram of CREATE is shown in Figure 3-12.

3.2.9.2 CREATE Screens

Informative message windows, as illustrated in Figure 3-13, are displayed to inform the user of the status of the simulation creation. If any errors are detected, an error message screen, as illustrated in Figure 3-14, is displayed that contains a pushbutton for the user to depress once the error messages have been comprehended.

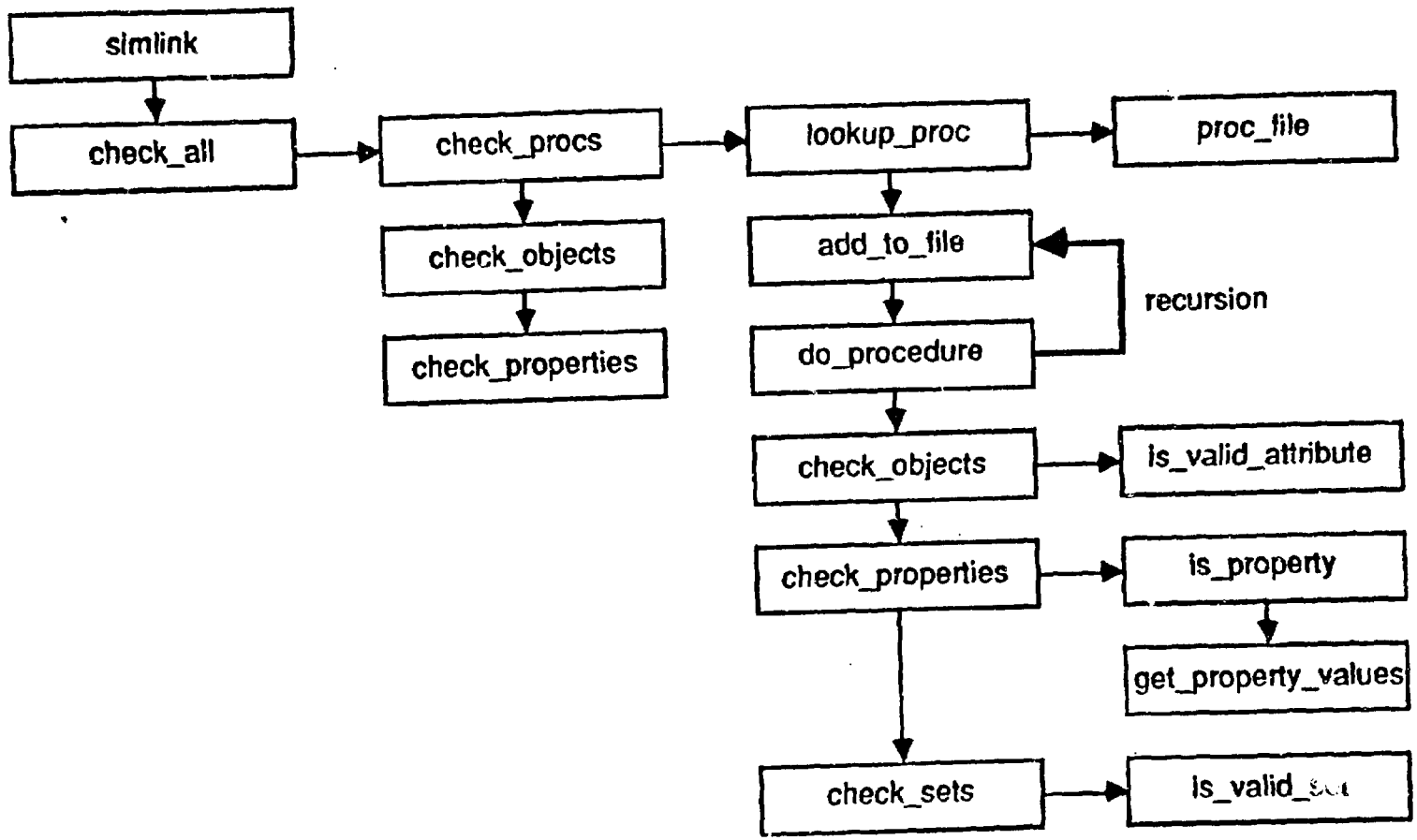


Figure 3-12. Simlink Functional Diagram

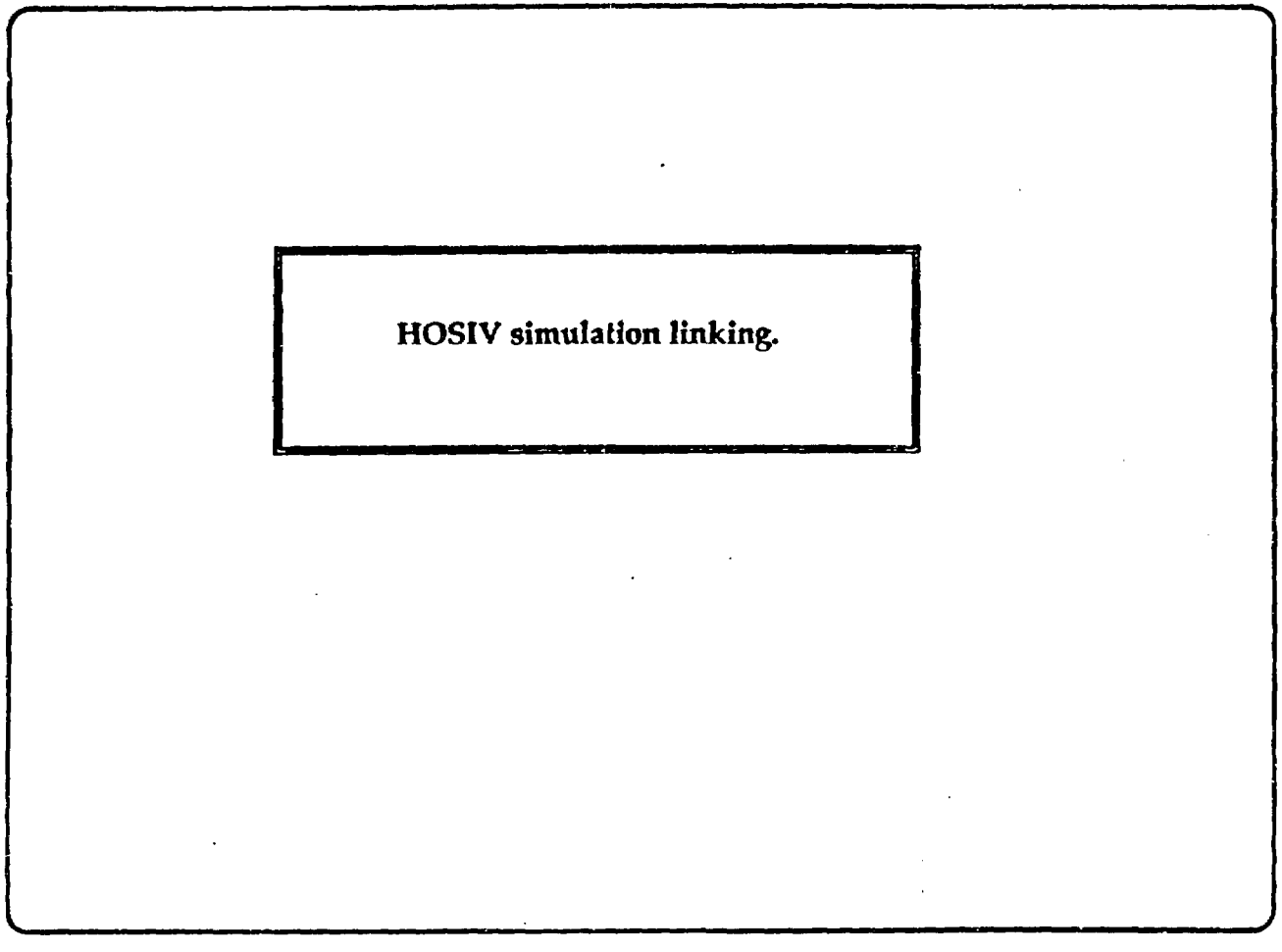


Figure 3-13. Link Message Window

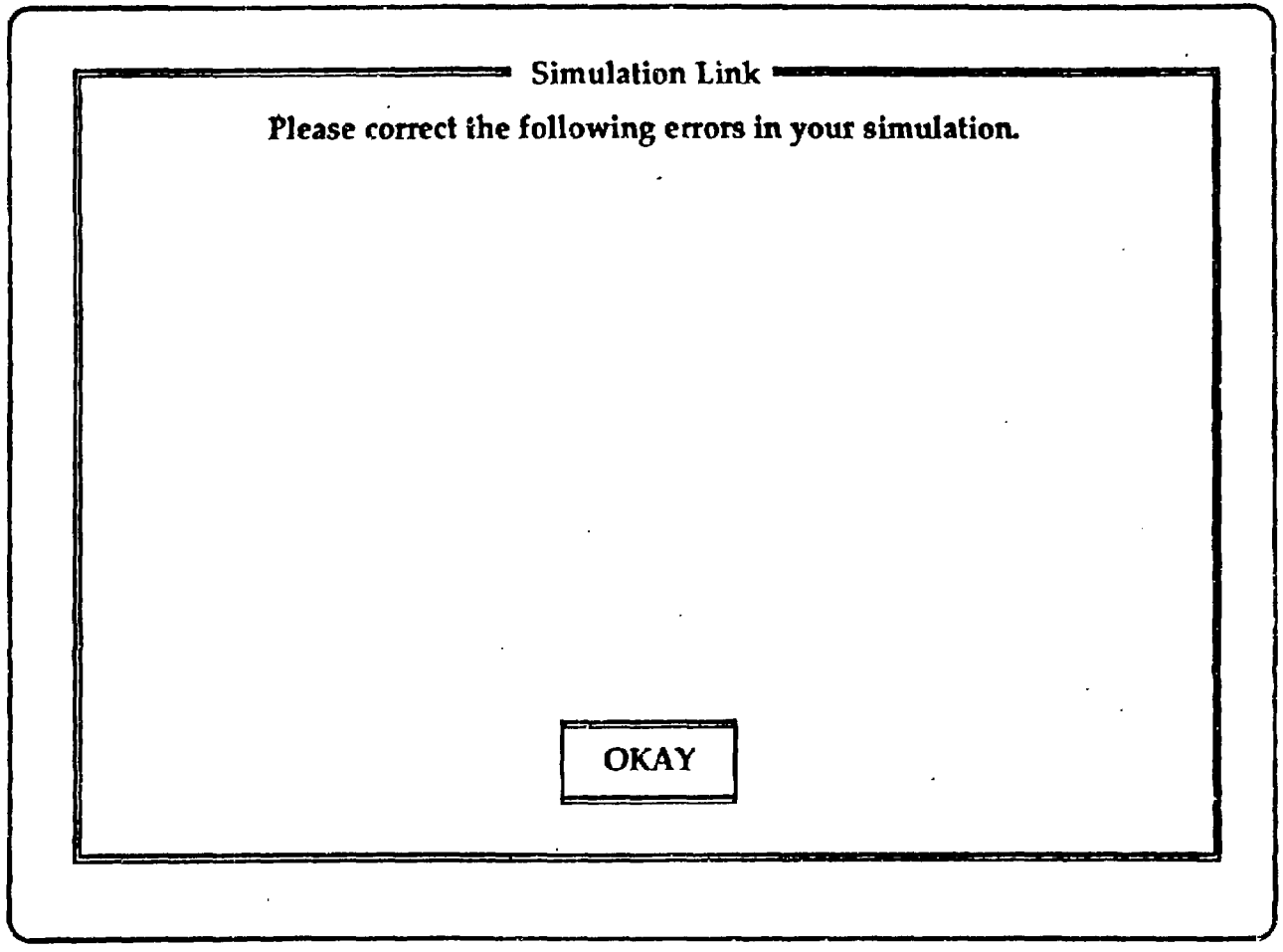


Figure 3-14. Link Errors Window

3.2.10 RUNSIM — Run Simulation

3.2.10.1 Description

RUNSIM executes the selected simulation and generates the simulation output files. It creates a screen showing the simulation status at each time increment and the currently active event, rule, and action.

3.2.10.2 RUNSIM Screens

The screen shown in Figure 3-15 allows the user to indicate whether the simulation is to start at the beginning, i.e., the start time specified in Simulation Setup, or restart the simulation at the point where it was previously terminated through the use of a **BEGINNING** and **RESTART** pushbutton. The run simulation screen, shown in Figure 3-16, shows the current status of the simulation through a series of labeled boxes as described below:

- simname — name of currently executing simulation.
- Current simulation time — the time of the current simulation in the form dd hh:mm:ss.ttt where dd is days, hh is hours, mm is minutes, ss is seconds and ttt is thousands of seconds.
- Action — the name of the action currently being processed by the simulation and the time the action started.
- Rule — the name of the rule currently invoked and the time the rule was triggered.
- Event — the name of the current event and the time the event started.

The screen also contains a **PAUSE** pushbutton that allows the user to interrupt the simulation. When the **PAUSE** button is depressed, the screen illustrated in Figure 3-17 is displayed that allows the user to indicate whether the simulation should continue or be terminated for later restart. If the user depresses the **CONTINUE** pushbutton, the simulation will resume execution; if the user depresses the **EXIT** pushbutton, the simulation will terminate. When the normal end of the simulation is reached, the screen displayed in Figure 3-18 will be displayed.

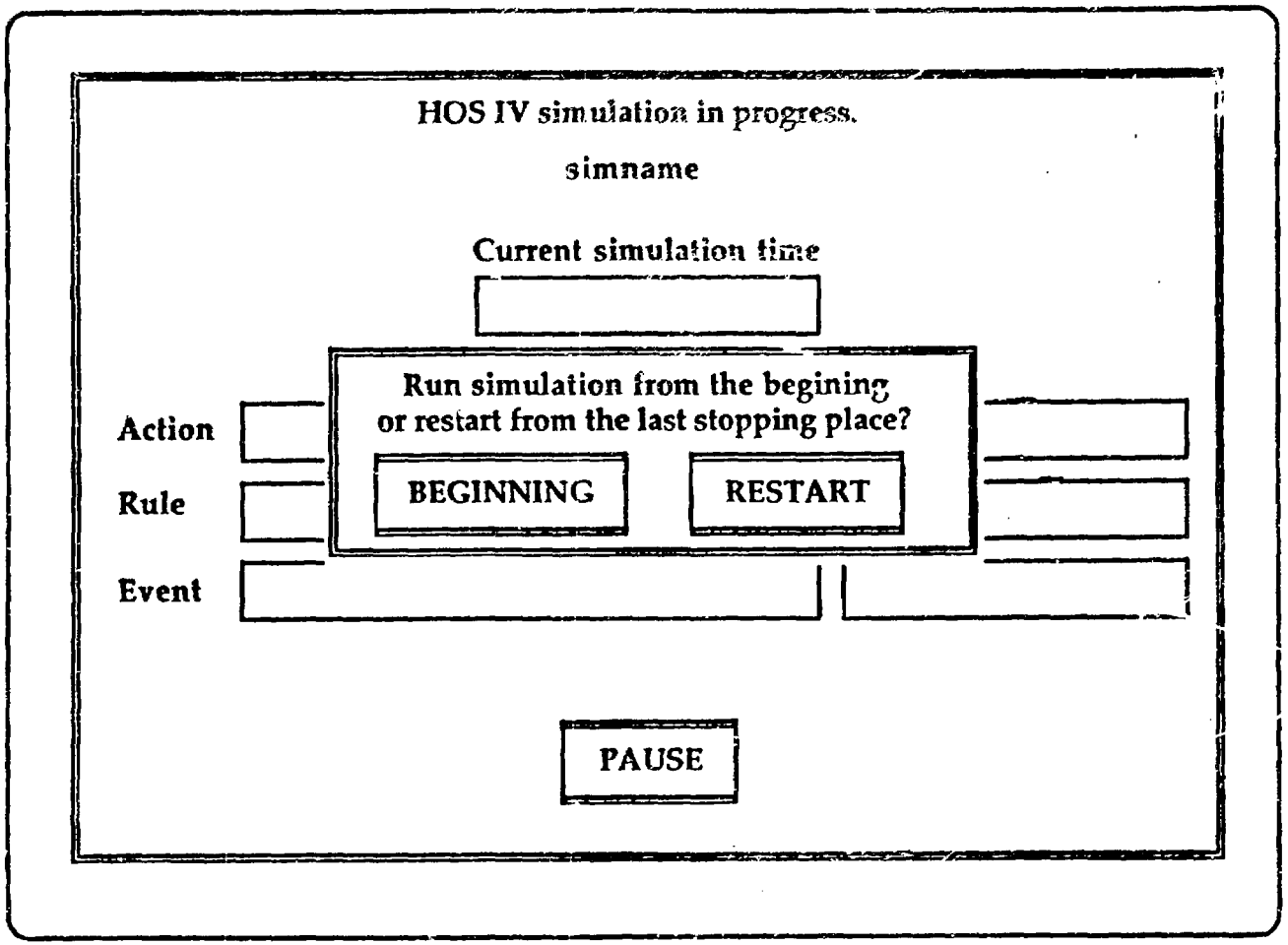


Figure 3-15. Beginning of Simulation Window

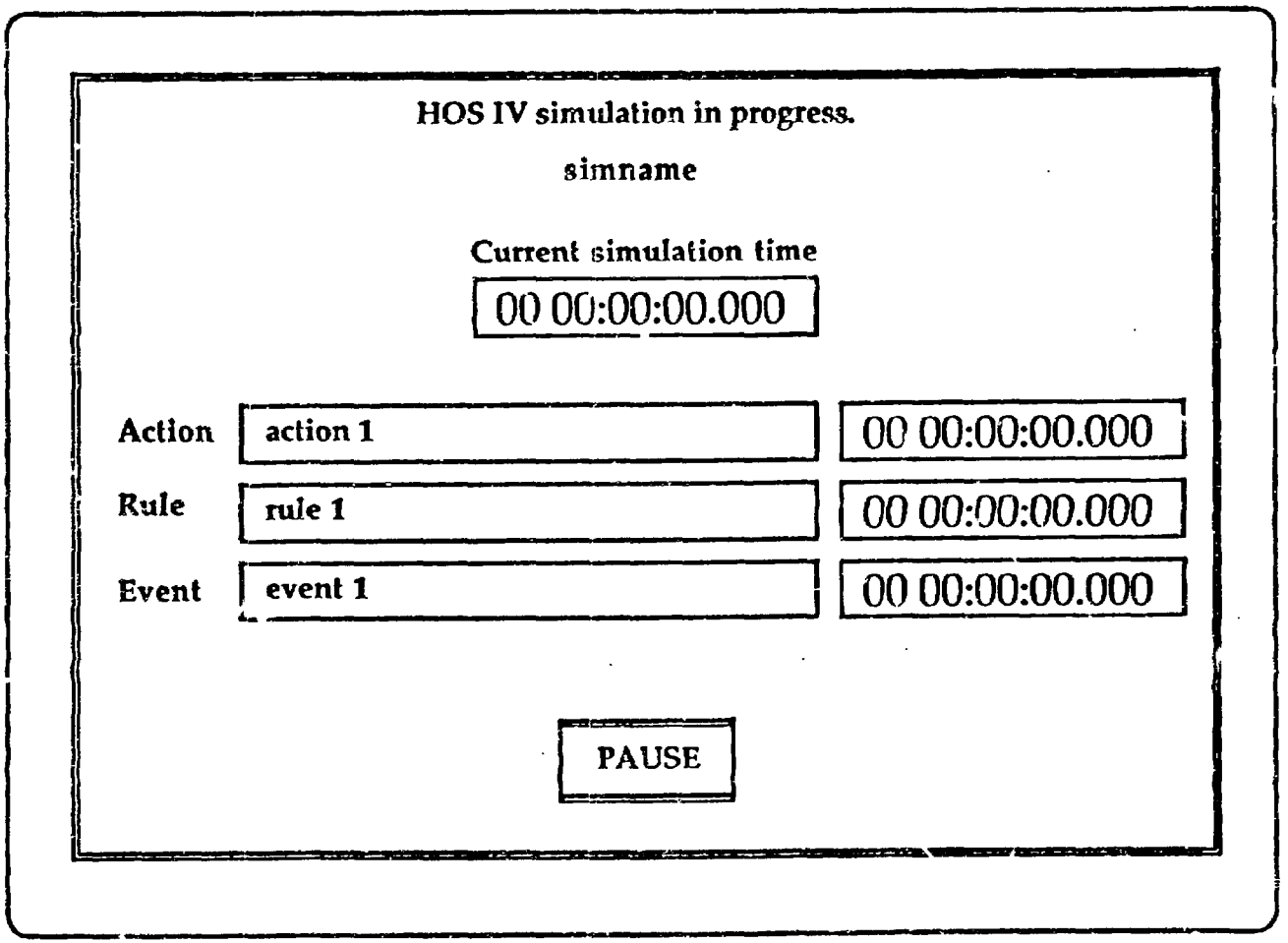


Figure 3-16. Simulation Window

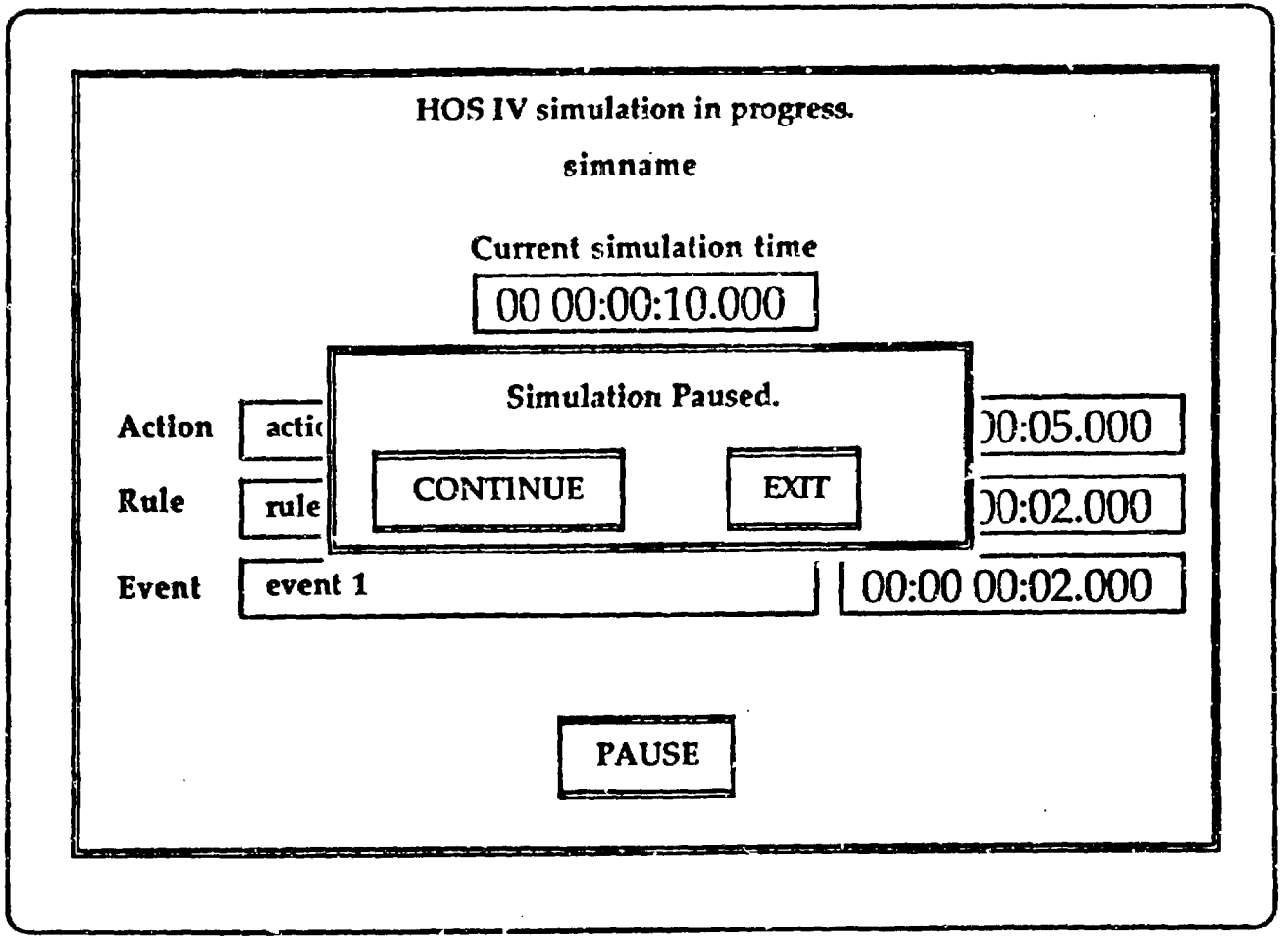


Figure 3-17. Simulation Paused Window

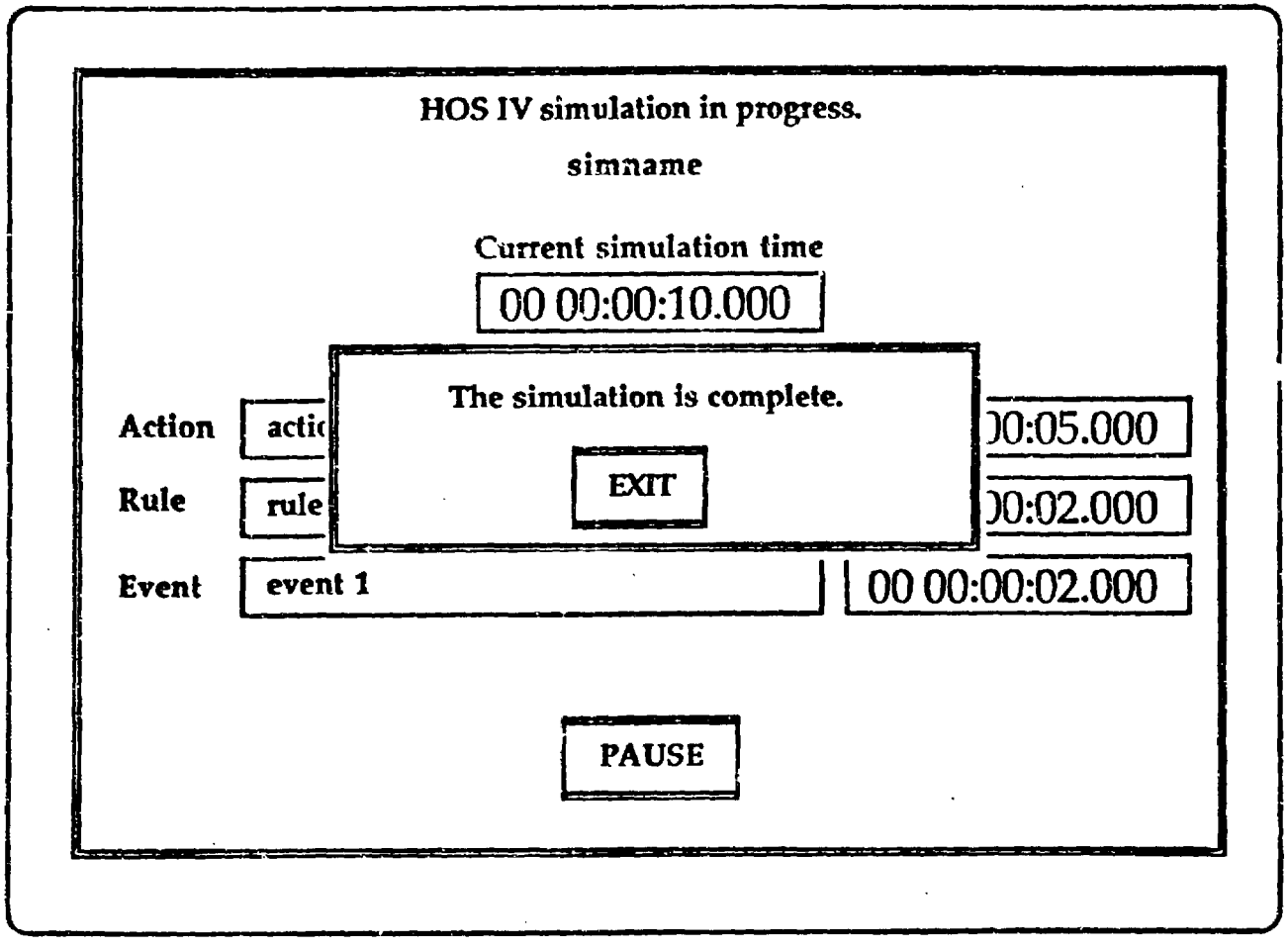


Figure 3-18. Simulation Complete Window

3.2.11 RESULTS — View Results

The View Results module is used to examine data produced by running a simulation. It is illustrated in Figure 3-19.

3.2.11.1 Description

RESULTS contains a series of standard reports that are used to assist in the analysis of simulation results. The available reports are the following:

- **Object Analysis** — contains the value of each object-characteristic pair at each time in the simulation when the value of the characteristic of the object changed.
- **Rule Analysis** — generates rule usage statistics including the number of times the rule was active and average duration of the activity.
- **User Simulation Output** — user-defined report produced by using the FILE verbs contained in actions.
- **Action Timeline** — generates a timeline showing the name of each active action at each time interval in the simulation.
- **Event Timeline** — generates a timeline showing the simulation time and event name of each active event.
- **Full Timeline** — combines the event, rule, action, and object timeline into one single report.
- **Object Timeline** — generates a timeline showing the simulation time when each characteristic of an object was modified.
- **Rule Timeline** — generates a timeline showing the simulation time and rule name of each active rule during the simulation.

3.2.11.2 RESULTS Screens

The View Results module consists of a title bar, a menu bar, a text viewing window with a scrollbar, and a number of dialog boxes used for program interaction with the user as illustrated in Figure 3-20.

View Results Module Title Bar. The title bar contains the words 'VIEW RESULTS' as the function name in the center. It does not use the current activity or status information areas of the title bar.

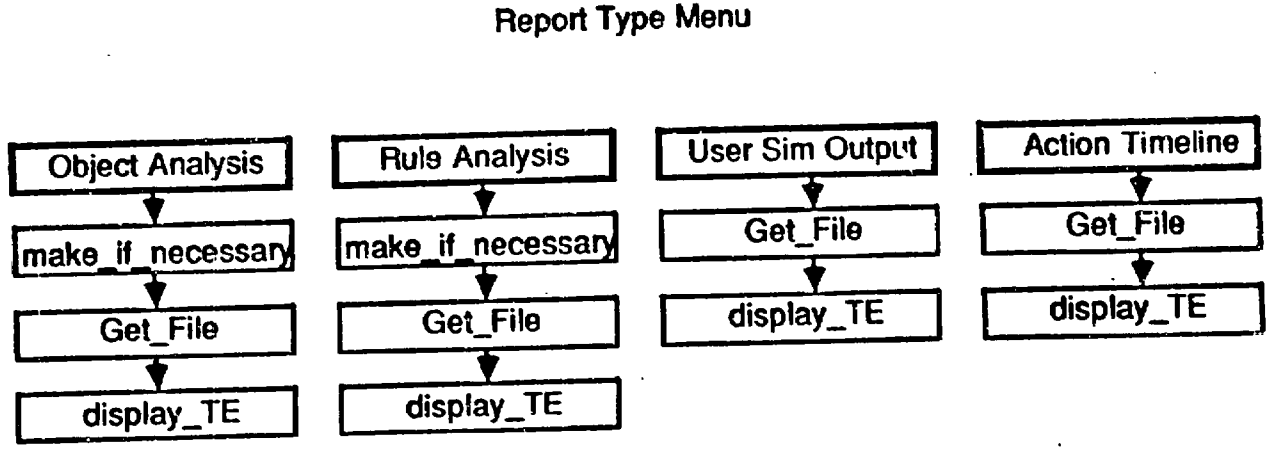


Figure 3-19. View Results Functional Diagram

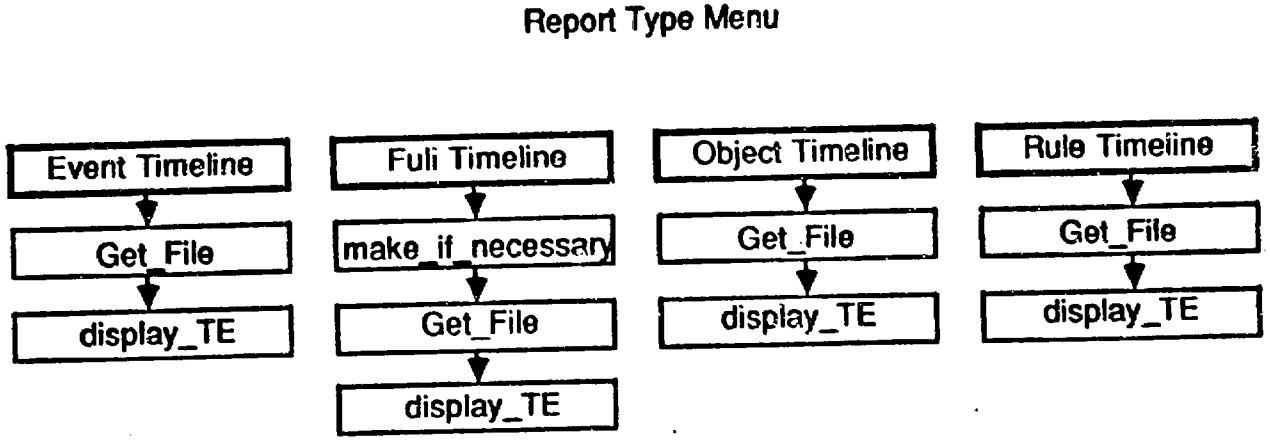


Figure 3-19. View Results Functional Diagram (continued)

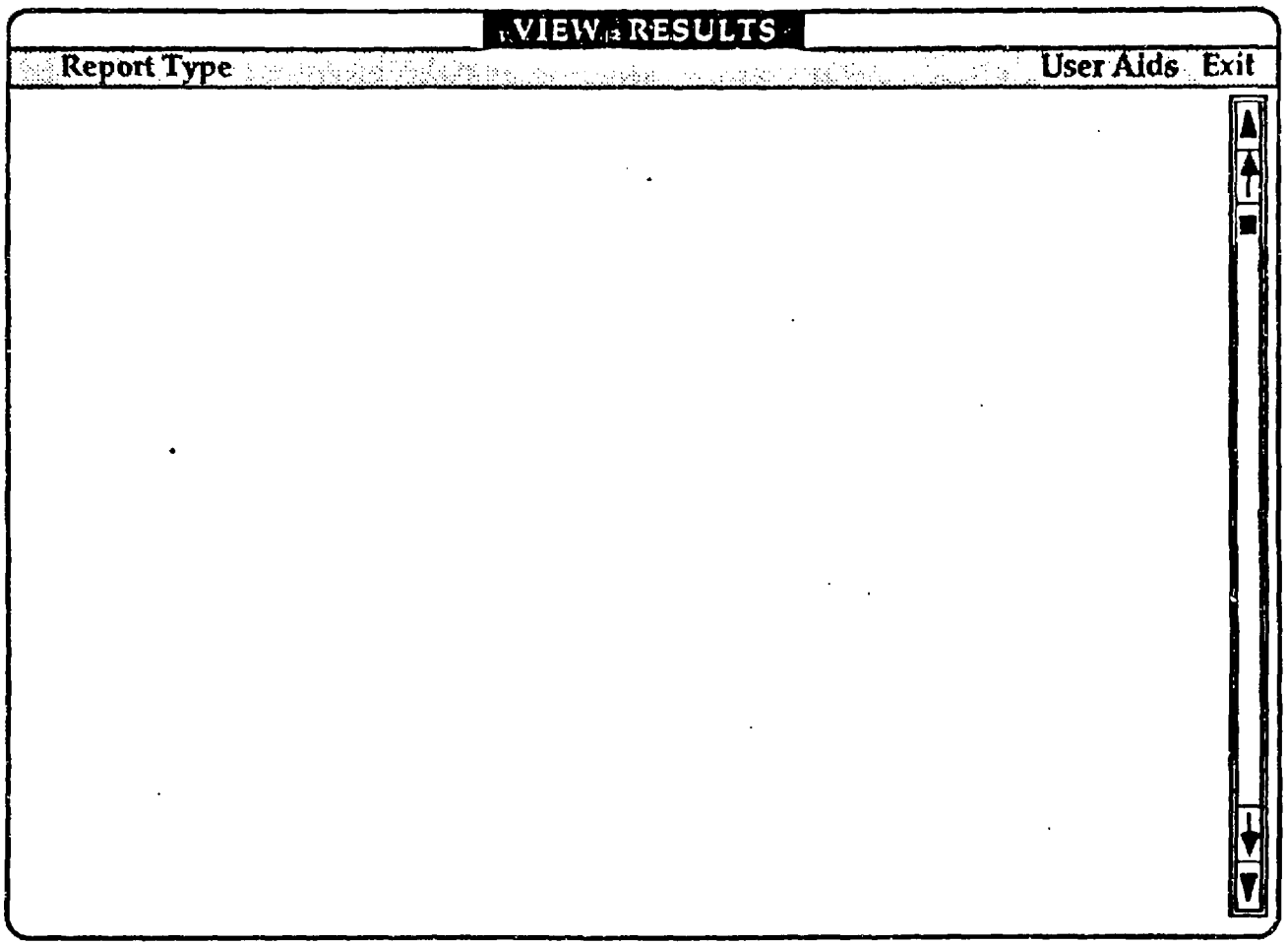


Figure 3-20. View Results Screen

View Results Module Menu Bar. The menu bar for the View Results module contains the following menu options:

- **Report Type** — file related commands such as saving, opening, etc.
- **User Aids** — provides the capabilities to view help files and action files.
- **Exit** — terminates View Results module and returns the user to the HOS-IV module.

The **Report Type** pull down menu contains:

- **Object Analysis** — opens the Object Analysis report; if it does not exist the create missing report message window is displayed.
- **Rule Analysis** — opens the Rule Analysis report; if it does not exist the create missing report message window is displayed.
- **User Simulation Output** — opens the User Simulation Output report; if it does not exist the report missing message window is displayed.
- **Action Timeline** — opens the Action Timeline report; if it does not exist the report missing message window is displayed.
- **Event Timeline** — opens the Event Timeline report; if it does not exist the report missing message window is displayed.
- **Full Timeline** — opens the Full Timeline report; if it does not exist the create missing report message window is displayed.
- **Object Timeline** — opens the Object Timeline report; if it does not exist the report missing message window is displayed.
- **Rule Timeline** — opens the Rule Timeline report; if it does not exist the report missing message window is displayed.

The **User Aids** pull down menu contains commands that allow the user to receive help on the current module and view action files.

- **Help** — allows the user to obtain additional information about using View Results module.
- **View File** — allows the user to view action files created using the View Results module.

View Results Module Text Viewing Window. The text viewing window consists of text area and a scrollbar. The text area is 23 rows by 77 columns.

The scrollbar is to the right of the text area and has four control buttons and a relative file position indicator. The four controls on the scrollbar are:

- **Scroll Line Down** — displays a page of text starting from the line before the current top line.
- **Scroll Page Down** — displays a page of text starting one page before the current top line.
- **Scroll Page Up** — displays a page of text starting from the current bottom line.
- **Scroll Line Up** — displays a page of text starting from the line after the current top line.

View Results Module Dialog Boxes. If the selected report requires additional information, text entry boxes are generated to request the appropriate information.

View Results Module Message Windows. Message windows display information which the user must acknowledge by depressing one of the selected pushbutton. Some message windows may be cancelled.

- **File not found** — the indicated file contains the simulation results and could not be located.
- **Create missing report** — the user is requesting output of a report that has not yet been generated.
- **End viewing session** — confirmation that the View Results function is terminating.

View Results Module Information Windows. Information windows require no input from the user. They are informative messages indicating that the system is carrying out the entered command and to let the user know the status of the operation.

- **Reading file** — the file is being read.

3.2.11.3 Input/Output

RESULTS uses the following files:

- **d:\htemp.txt** — temporary file created for saving and printing purposes.
- **c:\hosiv\hoshelp\vrt_help.000** — list of help topics specific to the View Results module.

- **c:\hosiv\hoshelp\vrt_help.nnn** — help files (numbered extension starting from 001).
- **c:\hosiv\cursim.dat** — name of the currently selected simulation.
- **c:\hosiv*cursim**cursim*.dob** — simulation log for object activity.
- **c:\hosiv*cursim**cursim*.dpr** — simulation log for action activity.
- **c:\hosiv*cursim**cursim*.dev** — simulation log for event activity.
- **c:\hosiv*cursim**cursim*.dtk** — simulation log for task activity.
- **c:\hosiv*cursim**cursim*.tim** — simulation timeline.
- **c:\hosiv*cursim**cursim*.log** — user-defined simulation output generated from FILE verbs.
- **c:\hosiv*cursim**cursim*.rul** — simulation rule information.
- **c:\hosiv*cursim**cursim*.oba** — simulation object information.

3.2.11.4 Maintenance Procedures

View Results module source code (viewrslt.c and vrt_menu.c) is compiled using Microsoft C version 4.0 with the large memory model switch (/AL). It requires the HOS-IV, SKYL, CT, and TE libraries in addition to the standard C libraries for linking.

4. HOS-IV FILES

4.1 Direction/Subdirectory Organization

All HOS-IV files are stored in a separate subdirectory —HOSIV — under the DOS root directory. The HOS-IV files are organized into the following subdirectories under HOSIV:

HOSC	Contains translated C code form action translator, the action editor, HAL translator, and HOS_LINK.EXE, and SIM.EXE.
HOSHPL	Contains HAL action code entered by the user through the action editor (EDIT__PRC.EXE). Used by the action editor and the HAL translator.
HOSINC	Contains one '#include' file of definitions for each file translated by HAL. Used by the HOS_LINK and Microsoft C compiler.
HOSOB	Contains one file for each translated action which lists the objects referenced in the action. Used by HOS_LINK to check for the existence of referenced objects in the object library.
HOSPRO	Contains one file for each translated action which lists the actions referenced by the action. Used by the Microsoft C compiler, in conjunction with SIM.C to produce the executable simulation code.
HOSPRP	Contains one file for each translated action which lists the alphabetics referenced in the action. Used by HOS_LINK to verify the existence of the alphabetics in the alphabetics library.
HOSSET	Contains one file for each translated action which lists the sets referenced within the action. Used by HOS_LINK to check for the existence of referenced sets in the object set library.
MSC	Contains all of the Microsoft C V4.0 compiler files and subdirectories. Used by HOS_LINK to create the simulation runtime module.
simname	Directory of files associated with the simulation named simname. Used by the rule editor, sim_set, and event editor.

4.2 File Descriptions

Appendix C contains a list of the HOS-IV files arranged by executables, data files, source code files, and batch files.

APPENDIX A
HOS-IV INDIVIDUAL PROGRAM UNIT DESCRIPTION

This appendix contains a listing of every function in all of the HOS-IV programs. The formal C prototype is given, followed by a brief description of what the function does. Functions are listed by source file in alphabetical order.

CUT_COPY.C

```
void begin_mark()
```

Sets the beginning of the selected text. If the end of the selected text has already been set, begin_mark calls highlight_text.

```
void clear_text()
```

Actually carries out the deletion of text. Called by do_clear.

```
int do_clear()
```

Deletes selected text from TE->text and throws it away.

```
int do_copy()
```

Copies selected text from TE->text and stores it in the clipboard. Old contents of the clipboard are destroyed.

```
int do_cut()
```

Deletes selected text from TE->text and stores it in the clipboard.

```
int do_paste()
```

Calls paste text to insert contents of clipboard in TE->text. If text is selected it first calls do_clear.

```
void end_mark()
```

Sets the end of the selected text. If the beginning of the selected text has already been set, end_mark calls highlight_text.

```
void exchange_pointers (num1, num2)
  long *num1, *num2;
```

Swaps ptr1 with ptr2.

```
void exchange_values (num1, num2)
  int *num1, *num2;
```

Swaps num1 with num2.

`void get_doc_text()`

Gets the text from TE->text for cut and copy.

`void highlight_text()`

Highlights text selected by begin_mark and end_mark.

`void init_clipboard()`

Empties out the clipboard.

`void paste_text()`

Inserts contents of clipboard at TE->insertion.

`void unhighlight_text()`

Unhighlights selected text.

```
void allocate_db_fields()
```

Walks the linked list of zones for the current window and allocates a text_edit record for each zone whose mode is not equal to SCREEN. It produces a circular linked list of text_edit records with the location of the head stored in the global pointer db_edit.

```
int db_mode (curr_zone)
   ZONE *curr_zone
```

Returns the mode of curr_zone based on the contents of curr_zone's message field.

```
int db_touch (arg)
   int arg;
```

Walks the linked list of text_edit records to determine which field of a dialog box was clicked in.

```
TEType *deallocate_db_fields (temp)
   TEType *temp;
```

This function deletes a circular linked list of text_edit records based on the global pointer db_edit. It is usually called with db_edit->next.

```
void Erase_db_field ()
```

This function erases all of the text in the current TE->text.

```
int extent_compare (b1, b2)
   box *b1, *b2;
```

Compares the size of b1 to b2 and returns TRUE if they are equal.

```
void extent_convert (temp)
   BOX *temp;
```

Converts temp from the format that Skylights stores it in (measured in half characters) to whole characters.

```
void get_pad_str (curr_len, max_len, pad_str, pad_char)
   int curr_len, max_len;
   char *pad_str, *pad_char;
```

Pads pad_str with copies of pad_char from the current length to its maximum length.

```
void print_db_str (my_num, max_len, row, col)
   int my_num, max_len, row, col;
```

Prints a numerical dialog box field string, max_len is the maximum length of the string field.

```
void str_to_db_field (string)
   char *string;
```

Inserts string in the text_edit record pointed to by the global pointer TE. It then sets TE equal to to the next record in the linked list.

```
void check_Time_error(evt)
   event *evt;
```

This routine compares the event time of the event passed to check_Time_error with the maximum simulation time. If the event time is too large, trap_error is called, otherwise Proc_error is called.

```
void clean_up()
```

This routine starts with the first event in the linked list and traverses the list filling in the event number field with successive integers.

```
char *cvt_evt_num(the_evt)
   event *the_evt;
```

This routine converts the event number to a three character leading zero string which it returns.

```
char *cvt_evt_time(the_evt)
   event *the_evt;
```

This routine converts the event time record to a number in terms of the minimum time unit. It then converts this number to a six character leading zero string which it returns.

```
void delete_evt(an_evt)
   event *an_evt;
```

Removes an_evt from the linked list of events and stores it in curr_evt, which is the event currently being worked on by the user.

```
void do_print()
```

This routine prints the entire linked list of events in a user-friendly format.

```
int evt_sel(line)
   int line;
```

This routine prompts the user if the current event has changed and then finish_sel is called to actually update the display.

```
void evt_to_window(evt)
   event *evt;
```

Displays the event passed to evt_to_window on the screen.

```
int explicit_save(arg)
   int arg;
```

This demon initializes curr_save_destination to point to the procedure is_verified and save_destination to point to finish_explicit_save. Then, it calls verify_evt and returns ok to skylights.

```
void fill_in_evt_rec(evt,line)
   event *evt;
   char line[120];
```

Fills in an element of the linked list of events.

```
event *find_evt(line_num)
   int line_num;
```

This routine returns the address of an event in the linked list based on its line number in the list box data structure. This is not a problem because they are both sorted.

```
void finish_explicit_save()
```

This routine does the appropriate list box updating after a save.

```
void finish_new_evt()
```

This routine does the linked list processing necessary for a new event.

```
void finish_sel()
```

This routine updates the list box display and the linked list of events.

```
int finish_setup_exit()
```

This routine opens the 'Are you sure you want to exit...' window.

```
char *get_list_line(event_ptr)
   event *event_ptr;
```

This routine returns the line that will appear in the list box window based on the event passed in. The line has the following format: dd:hh:mm:ss.xxx procedure description

```
void get_sim_info()
```

This function reads the simulation name, minimum time unit, and the maximum time into the appropriate variables.

```
event_time *get_time_record(time)
   char time[];
```

Inputs a time string of length 6 and outputs a time record pointer of type *event_time. An event time record consists of seven string fields (days, hours, minutes, seconds, tenths, hundredths, and thousandths).

```
char *get_time_str(num)
   int num;
```

This routine converts the number passed into a two character leading zero string and returns it.

```
void goto_save_destination()
```

This routine calls the routine pointed to by save_destination if save_destination is not NULL.

```
int handle_error(arg)
   int arg;
```

This routine closes the error window.

```
void init_demons()
```

initializes all of the demon function pointers for skylights.

```
event *init_evt()
```

This function returns a new event properly initialized.

```
void init_general()
```

This function initializes Text Edit globals.

```
void init_time_units()
```

Initializes time ratios array used to convert time in terms of the minimum time unit.

```
void init_windows()
```

Loads most of the windows used the in the Event Editor from the catalog 'edit_evt.cat', also opens those that will be immediately displayed.

```
void initialize()
```

This function calls all initialization routines such as init_windows, init_demons, init_time_units, get_sim_info, init_general, turn_off_unused_zones, and load_events, as well as initializing global variables.

```
void insert_evt(evt_to_insert)
   event *evt_to_insert;
```

This routine inserts the event record passed to it into the linked list of events.

```
void is_verified()
```

This routine sets curr_evt equal to the temporary evt used for the error checking now that it has been verified. It then calls goto_save_destination.

```
void load_events()
```

This function initializes the list box display with previously defined events if the call to read_events is successful. Otherwise, the user is put in 'new event mode'.

```
void main()
```

This function calls the main initialization routine and then starts the demon watcher.

```
int new(arg)
   int arg;
```

This demon calls save_if_necessary with the parameter finish_new_evt, and then returns ok to skylights.

```
event *new_evt()
```

This function attempts to return a pointer to an event record.

```
void Proc_error(evt)
   event *evt;
```

This routine sets the verify destination to be the current save destination and makes a call to Proc_file. If the procedure is undefined, setup_warning is called. Otherwise, the procedure pointed to by the verify destination is called.

```
int process_delete(arg)
   int arg;
```

This routine processes the answer to the 'Are you sure you want to delete...' prompt. It updates the screen and linked list as required.

```
int process_exit(arg)
   int arg;
```

This routine processes the user's answer to the 'Are you sure you want to exit...' prompt. If the answer is 'Yes', all of the necessary terminating routines are called. Otherwise, nothing happens.

```
int process_warning(arg)
   int arg;
```

This routine processes the answer to the warning prompt. If the answer is 'Continue', the procedure pointed to by verify destination is called. If the answer is 'Cancel', the event is not saved.

```
int read_events(filename)
   char filename[];
```

Attempts to open the file hosiv\simname\simname.ev1. If it exists, it reads in the previously defined events into a linked list data structure and returns TRUE. If the file does not exist, it returns FALSE.

```
int save_chg(arg)
   int arg;
```

This routine responds to the 'Save changes...' prompt. If the user selected 'Yes', curr_save_destination is set to is_verified and verify_evt is called. Otherwise, it calls goto_save_destination.

```
void save_evt()
```

This routine does the appropriate list box and linked list updating needed to save an event.

```
int save_if_necessary(function)
   func_ptr function;
```

This routine displays the Save changes window if any changes have occurred; otherwise, it calls goto_save_destination.

```
void set_save_destination(function)
   int (*function ) ();
```

Sets save_destination equal to parameter passed in.

```
int setup_delete(arg)
   int arg;
```

If there are any events to delete, this routine displays the "Are you sure you want to delete this event?" window.

```
int setup_exit()
```

This routine calls save_if_necessary with the function name finish_setup_exit.

```
void setup_warning(evt)
   event *evt;
```

This routine loads the warning window and displays it with the appropriate warning message.

```
void trap_error(evt)
   event *evt;
```

This routine loads the error window and displays it with the appropriate error message.

```
void turn_off_unused_zones()
```

Permanently destroys skylights pointers to demons for time zones that aren't needed based on the minimum time unit.

```
void verify_evt()
```

This routine gets the current event being displayed on the screen into temp_verify_evt. Then check_Time_error is called with this parameter.

```
void write_evt_files()
```

This routine writes out three files: simname.EV1, simname.EV2, and simname.EV3.

```
int exit_editor (arg)
   int arg;
```

Processes user input from the window which asks the users if they actually want to exit the Action Editor.

```
void goto_line ()
```

Finds the line specified in the find line number window. If the line number is greater than the total number of lines, the last line is found.

```
void goto_save_destination ()
```

This function executes save_destination. Save destination is a function pointer whose value depends on where the save was initiated.

```
void habort(message)
   char *message;
```

Ends the skylights demons, closes the windows, prints the last message, and spawns hosiv.exe.

```
void init_demons
```

Sets up the demon pointers.

```
void init_general ()
```

Sets up general text_edit stuff.

```
void init_windows()
```

Loads all the windows and opens the needed ones.

```
void initialize()
```

Calls all the appropriate initialization routines.

```
int invalid_name (the_str)
    char *the_str;
```

Checks the input action name for invalid characters and other errors.

```
void main ()
```

This function calls the main initialization routine, then starts the Skylights demon watcher.

```
int process_del_err1 (arg)
   int arg;
```

Processes user input from the window which says that you cannot delete a system file.

```
int process_del_err2 (arg)
   int arg;
```

Processes user input from the window which says that you cannot delete the current file.

```
int process_deletewin (arg)
   int arg;
```

Processes user input from the window which asks the user if the file should actually be deleted.

```
int process_error (arg)
   int arg;
```

Processes user input from the translation error message window.

```
int process_find_line (arg)
   int arg;
```

Processes user input from the find line number window.

```
int process_inputwin (arg)
   int arg;
```

Processes the user's response to the action name input window.

```
int process_inv_name_err (arg)
   int arg;
```

Processes user input from the invalid action name message window.

```
int process_not_found (arg)
   int arg;
```

Processes user input from the search string not found message window.

```
int process_ok (arg)
   int arg;
```

Processes user input from the translation successful message window.

```
int process_search (arg)
   int arg;
```

Processes user input from the input search string window.

```
int process_translator_feedback ()
```

Processes error messages and successful returns from the translator.

```
int quit (arg)
   int arg;
```

This function asks you if you want to save, then allows you to exit.

```
int read_file (arg)
   int arg
```

Processes user input from the action name selection window.

```
int save_changes (arg)
   int arg
```

Processes the user response to the dialog which asks the user if the changes should be saved.

```
int save_if_necessary (function)
   func_ptr function;
```

If the file has been changed, it asks the user if they want to save the last set of changes.

```
void search ()
```

Searches TE->text for the string specified in the input search string window. If the string is not found, it opens the string not found message window.

```
void set_save_destination (function)
   func_ptr function;
```

Sets the destination for the goto_save_destination function.

```
int setup_del_err1 ()
```

Opens the window which tells the user that he has tried to delete a file needed by the system.

```
int setup_del_err2 ()
```

Opens the window which tells the user that he has tried to delete the current file.

`int setup_deletewin ()`

Opens the window which asks the user if he actually wants to delete the file which they have selected.

`int setup_end ()`

Opens the window which asks the user if he actually wants to exit the Action Editor.

`int setup_filewin ()`

Opens the action name selection window.

`int setup_find_line ()`

Opens the find line number input window.

`int setup_inputwin ()`

Open the window used to get action names from the user.

`int setup_inv_name_err`

Opens the window which tells the user that there was an invalid name input and that he should try again.

`int setup_printwin`

Opens the printing in progress information window.

`int setup_search ()`

Opens the search string input window.

`int translate ()`

Spawns the HAL translator.

```
void cleanup ()
```

This function calls save_if_necessary with a pointer to finish_cleanup.

```
void fields_to_task (task)
   task_type * task;
```

This function takes the DB fields from the active window and puts them in the appropriate fields of the given task.

```
void finish_cleanup ()
```

This function writes, then deletes the task list, closes the window found in the global last_win, and deallocates the TextEdit records associated with it.

```
void goto_save_destiation ()
```

This function executes the function pointed to by the global function pointer save_destination after checking to be sure that save_destination is not NULL.

```
void habort (message)
   char *message;
```

This function exits the editor by spawning HOS-IV. It sends message to stdout. If the spawn fails, it will exit to DOS.

```
void init_demons ()
```

This function sets up all skylights demon names.

```
void init_error_check ()
```

This function initializes Task_bool.

```
void init_general ()
```

This function initializes global variables used by TextEdit.

```
void init_windows ()
```

This function opens the catalog, loads all windows, and opens the windows which are opened first.

```
void initialize ()
```

This function sets global flags and calls specific initialization functions.

```
void main ()
```

This function calls the main initialization routine, then starts the skylights demon watcher.

```
void make_new_task ()
```

This function makes a blank task and displays it for the user to edit.

```
int new (arg)
   int arg;
```

This function is called when the user presses the new button. It calls save_if_necessary with a pointer to make_new_task.

```
int print (arg)
   int arg;
```

This function prints a single task when the print button is pressed.

```
int process_delete (arg)
   int arg
```

This function processes the user's response to the question "do you want to delete this task?"

```
int process_exit (arg)
   int arg;
```

This function processes the "do you want to exit?" window.

```
int save (arg)
   int arg;
```

This function verifies and saves the task when the save button is pressed.

```
int save_changes (arg)
   int arg;
```

This function processes the user's response to the dialog which asks if he wants to save changes, by either verifying and saving the task or ignoring the save.

```
int save_if_necessary (function)
   func_ptr function;
```

This function asks the user if he wants to save changes if FileChange is TRUE, else it executes the function pointer.

```
void save_task ()
```

This function saves the valid task and updates the list box appropriately.

```
void select_task ()
```

This function activates and displays the task selected by the user.

```
int set_priority (arg)
   int arg;
```

This function processes the 4 buttons which allow the user to change the group and number.

```
void set_save_destination (function)
   int (*function) ();
```

This function sets the global function pointer equal to the function pointer passed in.

```
int setup_delete (arg)
   int arg
```

This function displays the current task in the list if necessary and asks the user if he wants to delete it.

```
void setup_exit ()
```

This function opens the "do you want to exit?" window.

```
void setup_window (name)
   char name[];
```

This function opens the operator, hardware, or environment window based on name; then tries to read the associated file. It displays the first task.

```
void task_name (task, str)
   task_type *task;
   char * str;
```

This function creates the string to be displayed in list box based on the given task.

```
void task_to_window (task)
   task_type * task;
```

This function takes a task record and displays it in the active window for editing.

```
int tsk_sel (arg)
   int arg;
```

This function calls save_if_necessary with a pointer to select_task as long as the user did not select the current task.

```
void check_if ()
```

This function checks the syntax of the if statement. If it finds an error or warning, it calls trap_error, else it calls the next error checking routine: check_proc.

```
void check_num ()
```

This function checks for a conflict in task numbers. If it finds one, it calls trap_error, else it calls the next error checking routine: check_if.

```
void check_proc ()
```

This function checks to see if the procedure exists. If it does not, it calls trap_error, else it calls the next error checking routine: check_until.

```
void number_exists (task)
   task_type *task;
```

This function checks the given task against the task list to see if one with that number is already in it. If the number exists, it sets errnum to 6 and returns TRUE.

```
void print_where ()
```

This function prints where a warning or error came from, either the if or the until condition.

```
int process_error (arg)
   int arg;
```

This function closes the error window when the user presses the button.

```
int process_warning (arg)
   int arg;
```

This function closes the warning window when the user presses a button, then continues the save if the user pressed continue.

```
void setup_error ()
```

This function loads the error window, puts the appropriate message in it, then displays it.

```
void setup_warning ()
```

This function loads the warning window, puts the appropriate message in it, then displays it.

`void trap_error ()`

This function displays problems as errors or warnings as appropriate.

`void verify_task ()`

This function copies the current window fields to a temp task, then calls the first of a series of verification routines.

FILE_IO.C

```
void Get_File (f_name, proc_name)
   char *f_name, *proc_name
```

Reads a file from the hard drive, converts all carriage returns and tabs.

```
char *get_r_block(start_pos, request_bytes)
   long start_pos, request_bytes;
```

Gets a block of text from the ramdrive starting from start_pos up to request_bytes or EOF whichever comes first. Mallocs space for the string and returns a pointer to it.

```
void init_file ()
```

Initializes a new TextEdit record and resets the file change flag.

```
int new_file ()
```

Deletes the current TextEdit record and calls init_file.

```
void replace_ch1_ch2 (the_str, search_chars,
   replace_char)
   char *the_str, *search_chars, *replace_char
```

Replaces every occurrence search_chars in the_str with replace_char.

```
void Update_Ram_Drive ()
```

Updates the ramdrive file with changes to TE->text.

```
void Update_Text ()
```

Gets a screenful of text from the ramdrive.

```
void Write_file (f_name)
   char f_name[];
```

Writes out TE->text to f_name.

HOSPROC.C

```
int add_proc_to_lib (item)
   PROCLIB *item;
```

Takes a procedure name, and adds it to the procedure library if it is new.

```
int Create_Proc_Item (item)
   PROCLIB *item;
```

Gives the procedure name, if it doesn't already exist. This routine will call the routine to create a new filename and add the item to the library in the appropriate place.

```
int Delete_Proc_File (name)
   char *name;
```

If the procedure is defined in the library, it is rewritten and marked for deletion. The item is not actually deleted until a Reorganize_Proc_Lib is done.

```
int get_next_proc_file (newfilename)
   char newfilename[];
```

Looks to see which file number in the sequence is next, and takes fill in numbers before going to the end of the list.

```
int insert_proclib_record (item)
   PROCLIB *item;
```

Called by add_proc_to_lib.

```
int is_valid_proc (name)
   char name[];
```

Returns true if the procedure has been defined -- used in the link hos utility.

```
int Proc_File (name, file)
   char name[];
   char file[];
```

If the procedure is defined, Proc_File returns the filename in the file parameter. It returns SUCCESS or FAILURE, depending on whether the procedure has been defined.

```
int Ramdrive_Proc_File ()
```

Creates the ramdrive version of the procedure library used for actual processing during run time.

```
int Reorganize_Proc_Lib ()
```

Deletes records deleted by the user.

```
int Rewrite_Proclib ()
```

This procedure is called during the reorganization process to write the new procedure library from the temporary file to the permanent file.

```
int sort_alphabetic_proc (flag)
   int flag;
```

Sorts the procedure library by proc name and puts it on the ramdrive.

```
int write_proc_item (item)
   PROCLIB *item;
```

Does the actual write the end of the file.

IO_TSK.C

```
void close_files ()
```

This function closes all 7 output files.

```
void consolidate_files ()
```

This function combines 5 sets of operator, hardware, and environment files into a single file.

```
void open_task_files ()
```

This function opens all 7 output files with paranoid error checking.

```
void print_tasks ()
```

This function steps through the task list, writes tk1, then prints it.

```
void print_tk1 ()
```

This function opens the file *.tk1 and prints it.

```
int read_tasks (filename)
   char *filename;
```

This function reads the editor's task file. It returns FALSE if the file is not found.

```
void three_files_to_one (one, two, three, target)
   char *one, *two, *three, *target;
```

This function take the three given files and puts them one after the other into the target file.

```
void write_c (task, stream)
   task_type *task;
   FILE *stream;
```

This function writes the actual C code for each task.

```
void write_environ_init (task, stream)
   task_type *task;
   FILE *stream;
```

This function writes the C file which initializes the environment tasks.

```
void write_hardware_init (task, stream)
   task_type *task;
   FILE *stream;
```

This function writes the C file which initializes the hardware tasks.

```
void write_header (stream)
   FILE *stream;
```

This function writes a HPL format header for the text only file.

```
void write_init (task, stream)
   task_type *task;
   FILE *stream;
```

This function calls the appropriate function to write operator, hardware, or environment initialization files.

```
void write_operator_init (task, stream)
   task_type *task;
   FILE *stream;
```

This function writes the C file which initializes the operator tasks.

```
void write_task_list ()
```

This function steps through the task list, writes the task to the editor's file and calls the other output routines.

```
void write_tasks ()
```

This function is the top level output routine.

```
void write_tk1 (task, stream)
   task_type *task;
   FILE *stream;
```

This function writes the task in HPL format.

LIBRARY.C

```
int copy_attr(source_attr, source_obj, dest_attr,
   dest_obj, obj_file, SIM_TIME)
   char *source_attr, *source_obj;
   char *dest_attr, *dest_obj;
   FILE *obj_file;
   long  SIM_TIME;

int copy_const (value, attribute, object, obj_file,
   SIM_TIME)
   double value;
   char *attribute;
   char *object;
   FILE *obj_file;
   long SIM_TIME;

int copy_local (local, attribute, object, obj_file,
   SIM_TIME)
   LOCAL *local;
   char *attribute;
   char *object;
   FILE *obj_file;
   long SIM_TIME;

int get (local, attribute, object)
   LOCAL *local;
   char *attribute;
   char *object;

int Set (local)
   LOCAL *local;
```

LIST_TSK.C

```
void delete_task (task)
   task_type *task;
```

This function deletes the given task from the list after first making sure the given task is valid.

```
void delete_task_list ()
```

This function deletes the entire list of tasks.

```
task_type* find_task (count)
   int count;
```

This function steps down the task list count times and returns that task.

```
void free_task (task)
   task_type *task;
```

This function frees the memory used by the given task.

```
task_type* find_task ()
```

This function creates a new task and sets each field to a default value.

```
void insert_task (task)
   task_type *task;
```

This function inserts the given task in the task list based on the string created by task_name.

```
task_type *new_task ()
```

This function allocates memory for a new task, sets the memory space to zeros and sets the pointers to NULL.

LOWLEVDB.C

```
double attfloat (attribute, object)
   char *attribute, *object;
```

Returns the value of a characteristic of an object as a double.

```
long attint (attribute, object)
   char *attribute, *object;
```

Returns the long value of the characteristic of the object passed in.

```
char *attrstr(attribute, object)
   char *attribute, *object;
```

Returns the string value of the characteristic of the object passed in.

```
int Close_and_Save_Object_File (filename)
   char *filename;
```

Closes and saves the ramdrive runtime version of a simulation objects library to the name in filename.

```
LOCAL *const (type)
   int type;
```

Places the constant of type 'type' into a local variable which it creates and returns a pointer to the variable. The LOCAL * is global and is reused.

```
LOCAL *Create_Local_Variable (Type, local)
   unsigned char Type;
   LOCAL *local;
```

If previously uninitialized, it creates a local variable of type Type.

```
int Get_Attribute (attribute, object, local)
   char *attribute;
   OBJECTS *object;
   LOCAL *local;
```

Gets the value of the indicated characteristic (attribute) of the object and passes it back in local.

```
int Get_Object (object_name, object)
   char *object_name;
    OBJECTS *object;
```

Reads the directory, finds the object, reads the library and returns the object in object.

```
int Init_Object_File ()
```

Checks for the existence of the object file and opens it.

```
int is_valid_attribute (attribute, object)
   char *attribute, *object;
```

Returns TRUE or FALSE depending on whether the characteristic exists in the object.

```
int is_valid_object (object)
   char object;
```

Returns TRUE or FALSE depending on whether the object has been defined.

```
int Kill_Local_Variable (local)
   LOCAL *local;
```

Frees memory allocated for a local variable -- currently not being used.

```
double locfloat (local)
   LOCAL *local;
```

Returns the value of local as a double.

```
long locint (local)
   LOCAL *local;
```

Returns the long value of the local passed int.

```
int log_err (string )
   char *string;
```

Used for error reporting -- can receive any permissible format string allowed in printf, including variables.

```
int rewrite_item (position, item)
   int position;
   OBJECTS *item;
```

Rewrites an item that already exists.

```
int Set_Attribute(object, attribute, local, obj,
   obj_file, SIM_TIME)
   OBJECTS *object;
   char *attribute;
   LOCAL *local;
   int obj;
   FILE *obj_file;
   long SIM_TIME;
```

Sets the characteristic of the object to the value in local.

```
int Set_Attribute_To_Const (object, attribute, value,
   obj, obj_file, SIM_TIME)
   OBJECTS *object;
   char *attribute;
   double value;
   int obj;
   FILE *obj_file;
   long SIM_TIME;
```

Sets an attribute to the value contained in the double value.

```
unsigned int typeatt (attribute, object)
   char *attribute, *object;
```

Returns the type of the given characteristic.

```
double V(local)
   LOCAL *local;
```

Unused currently.

```
int write_item (item)
   OBJECTS *item;
```

Writes the item at the end of the file.

OBJDIR.C

```
int add_object_to_lib (item, flag)
   OBJECT *item;
   int flag;
```

Adds object to end of library.

```
int compare_obj(string1, string2)
   char *string1, string2;
```

Used by the sort_alphabetic_obj routine.

```
int create_object_directory (from_file)
   char *from_file;
```

Reads objects library and creates a ramdrive directory of object names.

```
int find_duplicate_object (item)
   char *item;
```

Returns SUCCESS if the object already exists, FAILURE if it doesn't.

```
int find_object_offset (item)
   char *item;
```

Given the name of the object, it returns the offset for seeking in the library.

```
int get_object_name (position, item)
   int position;
   char *item;
```

Returns the object name given the position.

```
int get_object_name_sorted (position, item)
   long position;
   char *item;
```

Returns the 'positionth' name from the sorted library listing.

```
int reorganize ();
```

Rewrites the objects library, deleting items deleted by the user.

```
int sort_alphabetic_obj (flag)
   int flag;
```

Sorts the directory onto the ramdrive.

OBJPROP.C

```
int add_item_to_dictionary (item)
   MEMBER *item;
```

Adds an alphabetic to the end of the dictionary.

```
char *build_property_dictionary (fromfile)
   char *fromfile;
```

Builds the ramdrive alphabetics dictionary.

```
int compare_prop (string1, string2)
   char *string1, *string2;
```

Used by the sort_alphabetic_dictionary for sorting.

```
int find_duplicate_property (item)
   char *item;
```

Checks to see if the property is already defined.

```
int get_next_name_sorted (position, yourstring)
   int position;
   char *yourstring;
```

Returns the 'position' name off the ramdrive file and returns it.

```
int get_property_name (position, yourstring)
   int position;
   char *yourstring;
```

Given the numeric value of the property, returns the name.

```
int get_property_value (item)
   char *item;
```

Given the string equivalent, returns the numeric value.

```
char *loc_char (local, yourstring)
   LOCAL *local;
   char *yourstring;
```

Returns the name of a property contained in a local variable.

```
int sort_alphabetic_proc (flag)
   int flag;
```

Sorts the alphabetics and puts them in a ramdrive file.

OBJSET.C

```
SET *Define_Object_Set(Set, setname)
   SET *Set;
   char *setname;
```

Assign the setname to variable Set and return pointer to Set.

```
int get_current_member (Set, member)
   SET *Set;
   char *member;
```

Returns name of the current member of set Set.

```
int get_first_member (Set, member)
   SET *Set;
   char *member;
```

Returns name of first member of Set.

```
int get_last_member (Set,member)
   SET *Set;
   char *member;
```

Returns name of the last member of set Set.

```
char *Get_Member (filename, which)
   char *filename;
   int which;
```

Actually reads the name of the member which.

```
int get_next_member (Set, member)
   SET *Set;
   char *member;
```

Returns next member of set.

```
int get_previous_member (Set, member)
   SET *Set;
   char *member;
```

Gets the member just before the current member.

```
int get_this_member (Set, member, this)
   SET *Set;
   char *member;
   int this;
```

Returns the 'this' member of the set.

```
int is_end_of_set (Set)
   SET *Set;
```

Returns TRUE if no more members sequentially in set, and FALSE otherwise.

```
char *make_member_name (name, which)
   char *name;
   int which;
```

Creates member name given number and setname.

```
int valid_member (member)
   char *member;
```

Returns TRUE if the set exists, FALSE if not.

```
int add_set_to_lib (SETLIB *)
```

Adds set to library.

```
int count_items_in_set (setname)
   char setname[];
```

Returns number of members in set.

```
int Create_Set_Item
   SETLIB *item;
```

If a set item is new, it creates the file name and inserts it into the library, else simply returns the set's information.

```
int delete_appropriate_objects (setname, number)
   char setname[];
   int number;
```

After marking the setlib record for deletion, goes and marks the sets objects for deletion.

```
int Delete_Set_File (name)
   char name[];
```

Deletes a setfile entry and its associated objects.

```
int get_next_set_file(newfilename)
   char newfilename[];
```

Creates a unique runtime file name for the set.

```
int get_set_name (which, name)
   int which;
   char name[];
```

Returns the nth set name.

```
int is_valid_set (name)
   char name[];
```

Returns TRUE for a valid set, FALSE for an invalid set name, where invalid is undefined.

```
int Reorganize_Set_Lib ()
```

Deletes records marked for deletion, and writes out the new set library.

```
int Rewrite_Setlib ()
```

Rewrites an existing entry in the set library.

```
Ramdrive_Set_File ()
```

Creates the setfile library ("HOSSET.S$) on the ramdrive for use in the Object Editor.

```
int Set_Entry_Position (name, file, items)
   char name[];
   char file[];
   int *items;
```

Returns Fseek position.

```
Set_File (name, file, items)
   char name[];
   char file[];
   int *items;
```

If the set exists, this routine returns the filename designation and number of members.

```
int write_set_item (item)
   SETLIB *item;
```

Appends a set entry to the end of the library.

TSK_MENU.C

```
int menu (arg)
   int arg;
```

This function dispatches all menu selections based on arg.

APPENDIX B
HAL ERROR MESSAGES

Error Message	Module Name	Description
-2	BOOL_PART	Incomplete Boolean clause.
-1	GET_NEXT_TOKEN	Action is not terminated with an END statement.
0	DO_GROUP	Normal EOF reached.
1	ADD_TO_TABLE	Internal HOS error. Dynamic memory requirements exceed those available. Consult HOS expert.
5	DO_KEYWORD	Looking for a HOS keyword and none was found.
6	DO_IT	Invalid HOS statement.
7	DO_IF	IF statement contains an invalid Boolean clause.
7	DO_IF	IF statement must be followed by THEN.
8	DO_IF	Invalid IF group block of statements. An IF block must contain an IF Boolean condition THEN ENDIF.
10	DO_IF	Invalid ELSE group block of statements. An ELSE block must follow an IF — ENDIF block. An ELSE statement must be terminated with an ENDELSE.
12	DO_GROUP	An IF, WHILE, or ELSE group contains an invalid statement.
18	DO_DEFINE	The define block contains an invalid type.
18	A NEXT_TOK	SET statement contains an invalid formula.
19	A NEXT_TOK	SET statement contains an invalid mathematical function.
22	BOOL_PART	Invalid Boolean clause.
24	STATEMENT	HOS statements must being with a HOS verb.
30	DO_WHILE	A WHILE statement contains an invalid Boolean clause.
30	UPDATE_TYPE	Internal HOS error in DEFINE — Number of defined variables exceeds maximum permitted in program. See HOS expert.
31	DO_WHILE	WHILE statement must be terminated with the word THEN.
31	UPDATE_TYPE	Internal HOS error in DEFINE — Problem with token -- consult an HOS expert.
32	DO_WHILE	Invalid WHILE group of statements. A WHILE is in the form WHILE Boolean condition THEN ... ENDWHILE.
32	UPDATE_TYPE	Internal HOS error in DEFINE — Invalid type. Consult an HOS expert.
33	DO_GROUP	An IF, WHILE, or ELSE group of statements is not terminated correctly with ENDIF (for IF block), ENDWHILE (for WHILE block), or ENDELSE (for ELSE block).
57	DO_C_CODE	END_C_CODE does not start in column one to terminate a block of user C code.
58	DO_C_CODE	END_C_CODE does not start in column one to terminate a block of user C code.
59	DO_C_CODE	A block of user defined C code must be terminated with an END_C_CODE statement. None was found.

Error Message	Module Name	Description
65	CNVT_VALUE	Internal HOS error — Unable to determine variable type. Consult an HOS expert.
66	CNVT_VALUE	SET formula contains an alphabetic variable. Only WHOLE and DECIMAL variables can be used in formulas.
67	CNVT_VALUE	SET statement cannot contain a object set name.
68	CNVT_VALUE	SET formula cannot contain the name of a local object.
70	DO_NEXT	Invalid token type.
71	DO_NEXT	Invalid token type.
72	DO_NEXT	Invalid or unexpected HOS keyword.
72	DO_NEXT	***Undefined ERROR code ***
73	DO_NEXT	Undefined variable. All local variables must be defined in the DEFINITIONS block.
73	DO_NEXT	A local variable must be a WHOLE, DECIMAL, or ALPHABETIC.
78	DO_NEXT	Unable to locate an object or local object name.
79	CNVT_VALUE	Invalid constant. Constants must be either WHOLE or DECIMAL number.
82	BOOL_PART	A Boolean clause contains a variable which is not an ATTRIBUTE, WHOLE, DECIMAL, or ALPHABETIC.
93	NEXT_TOK	SET statement formula contains a local variable that has not a WHOLE or DECIMAL variable.
100	DO_FORM	SET statement contains an incomplete formula. Check to make sure all parentheses are paired correctly.
100	NEXT_TOK	SET statement contains an incomplete formula. Check to make sure all parentheses are paired correctly.
101	NEXT_TOK	SET statement formula contains an undefined variable.
103	DO_FORM	SET statement formula contains an invalid operator, variable, or mathematical function.
154	DO_FORMULA	SET statement contains invalid or missing parenthesis.
155	DO_FORMULA	A SET statement contains unmatched or missing parenthesis.
200	CNVT_VALUE	Undefined local variable.
210	DO_FORMULA	SET statement contains different number of right and left parenthesis.
300	DO_START	Invalid START statement. START verb must be followed by task number, group name, or the words: HARDWARE, OPERATOR, ENVIRONMENT, or ALL.
300	DO_STOP	Invalid STOP statement. STOP verb must be followed by task number, group name, or the words: HARDWARE, OPERATOR, ENVIRONMENT, or ALL.

Error Message	Module Name	Description
301	WHICH_TASK	START, STOP, or SUSPEND verb contains an invalid task number.
302	DO_START	START statement is invalid.
302	DO_STOP	STOP statement is invalid.
302	DO_SUSPEND	SUSPEND statement is invalid.
303	WHICH_TASK	START, STOP, or SUSPEND verb contains an invalid task number.
330	DO_PUT	PUT statement is invalid.
331	DO_PUT	PUT statement contains a variable that is not an ATTRIBUTE, WHOLE, DECIMAL, or ALPHABETIC.
378	WHICK-TASK	START, STOP, or SUSPEND statement contains an invalid group number.
379	WHICH_TASK	START, STOP, or SUSPEND statement contains a group number that is less than 0 or greater than 9.
450	DO_PRINT	PRINT statement contains an invalid operator or is improperly formed.
451	DO_FILE	FILE statement is invalid.
451	DO_FILE	FILE statement contains a reserved HOS keyword.
451	DO_PRINT	PRINT statement contains a reserved HOS keyword.
501	DO_DEFINE	DEFINE statement group contains an invalid keyword.
502	DO_DEFINE	DEFINE block contains an invalid operator.
503	DO_RECEIVE	RECEIVE statement is invalid.
503	DO_USING	USING statement contains an invalid operator or invalid type.
550	DO_RECEIVE	RECEIVE verb contains a reserved HOS keyword.
550	DO_USING	USING statement contains a HOS keyword.
551	DO_RECEIVE	RECEIVE verb contains an invalid operator or variable.
551	DO_RECEIVE	RETRIEVE verb can only be used for SET objects.
551	DO_USING	USING statement contains an undefined variable.
555	DO_RECEIVE	RECEIVE verb parameters must be defined as either WHOLE, DECIMAL, LOC_OBJECT, or ALPHABETIC.
555	DO_USING	USING statement contains a parameter that is not a WHOLE, DECIMAL, OBJECT, LOC_OBJECT, or ALPHABETIC.
563	DO_RECEIVE	RETRIEVE statement contains an invalid variable.
565	DO_RECEIVE	RETRIEVE verb object set pointer must be defined as either a WHOLE or DECIMAL local variable.
566	DO_RECEIVE	RETRIEVE verb object set keyword must be either FIRST, LAST, NEXT, PREVIOUS, or CURRENT.
587	DO_SEED	SEED statement contains an undefined local variable.

Error Message	Module Name	Description
588	DO_SEED	SEED statement argument must be defined as either a WHOLE or DECIMAL variable or a numeric value.
589	DO_SEED	SEED statement is invalid.
610	DO_ATTRIBUTE	Internal HOS error — unable to determine attribute type. Consult an HOS expert.
610	DO_FILE	Internal HOS error in FILE statement — unable to determine local variable type. Consult an HOS expert.
610	DO_GET	Internal HOS error in GET statement — unable to determine local variable type. Consult an HOS expert.
610	DO_PRINT	Internal HOS error in PRINT statement — unable to determine local variable type. Consult an HOS expert.
610	DO_PUT	Internal HOS error with PUT statement — unable to determine local variable type. Consult an HOS expert.
610	DO_RETRIEVE	Internal HOS error in RETRIEVE statement — unable to determine local variable type. Consult an HOS expert.
610	DO_SET	Internal HOS error in SET statement — unable to determine local variable type. Consult an HOS expert.
610	DO_SET	Internal HOS error in SET statement— unable to determine variable type. Consult an HOS expert.
611	DO_SET	A SET statement contains a variable/value that is not WHOLE, DECIMAL, or ALPHABETIC.
700	DO_COMMENT	COMMENT statement contains a alphanumeric string that causes the HAL program to abnormally terminate.
789	PROCESS_TOKEN	Invalid use of hyphen/minus sign.
800	DO_RETRIEVE	RETRIEVE verb must contain the name of an object that has been defined as an LOC_OBJECT.
1000	HAL	Unable to open error.lst file.
1000	HAL	Internal HOS error — unable to open error.lst file. Consult an HOS expert.
1001	OPEN_FILE	Internal HOS error — unable to open input HAL file. Consult an HOS expert.
1002	OPEN_FILE	Internal HOS error — unable to open C file. Consult an HOS expert.
1003	OPEN_FILE	Internal HOS error — unable to open file containing action names. Consult an HOS expert.
1004	OPEN_FILE	Internal HOS error — unable to open file containing names of alphabetics. Consult an HOS expert.
1005	OPEN_FILE	Internal HOS error — unable to open file containing object names. Consult an HOS expert.
1006	OPEN_FILE	Internal HOS error — unable to open file containing names of variable definitions. Consult an HOS expert.
1007	OPEN_FILE	Internal HOS error — unable to open file containing set names. Consult an HOS expert.

Error Message	Module Name	Description
1008	HAL	Internal HOS error — cannot create symbol table. Consult an HOS expert.
2001	HAL	Invalid HAL keyword (NOTENDOFSET, ENDOFSET) in a conditional statement.

APPENDIX C
HOS-IV FILE DESCRIPTIONS

This appendix contains an alphabetical listing of all files used in HOS-IV. It is divided into three sections: Executables, Catalogs, and data files.

EXECUTABLES

EDIT_EVT.EXE

Location: C:\HOSIV

Spawned by: HOSIV.EXE

Event Editor executable code.

EDIT_OBJ.EXE

Location: C:\HOSIV

Spawned by: HOSIV.EXE

Object Editor executable code.

EDIT_PRC.EXE

Location: C:\HOSIV

Spawned by: HOSIV.EXE

Action Editor executable code.

EDIT_TSK.EXE

Location: C:\HOSIV

Spawned by: HOSIV.EXE

Rule Editor executable code.

HOS_LINK.EXE

Location: C:\HOSIV

Spawned by: HOSIV.EXE

Create simulation executable code.

HOSIV.EXE

Location: C:\HOSIV

Spawned by: RUNHOS.BAT

Main HOS executable.

HPL.EXE

Location: C:\HOSIV

Spawned by: EDIT_PRC.EXE

Action translator executable code.

OBJANAL.EXE

Location: C:\HOSIV

Spawned by: VIEWRSLT.EXE

Object analysis report program.

SETUPSYS.EXE

Location: C:\HOSIV

Spawned by: INSTALL.BAT

System setup executable.

SIM.EXE

Location: C:\HOSIV

Spawned by: HOSIV.EXE

The current simulation's executable code.

SIMSET.EXE

Location: C:\HOSIV

Spawned by: HOSIV.EXE

Simulation setup executable code.

TASKANAL.EXE

Location: C:\HOSIV

Spawned by: VIEWRSLT.EXE

Rule analysis report program.

TIMELINE.EXE

Location: C:\HOSIV

Spawned by: VIEWRSLT.EXE

Timeline analysis report program.

VIEWRSLT.EXE

Location: C:\HOSIV

Spawned by: HOSIV.EXE

Simulation result analysis program executable code.

CATALOGS

EDIT_EVT.CAT

Location: C:\HOSIV

Used by: EDIT_EVT.EXE

Catalog of windows used by the Event Editor executable.

EDIT_OBJ.CAT

Location: C:\HOSIV

Used by: EDIT_OBJ.EXE

Catalog of windows used by the Object Editor executable.

EDIT_PRC.CAT

Location: C:\HOSIV

Used by: EDIT_PRC.EXE

Catalog of windows used by the Action Editor executable.

EDIT_TSK.CAT

Location: C:\HOSIV

Used by: EDIT_TSK.EXE

Catalog of windows used by the Rule Editor executable.

HOSIV.CAT

Location: C:\HOSIV

Used by: HOSIV.EXE

Catalog of windows used by the HOS-IV executable.

LINK.CAT

Location: C:\HOSIV

Used by: HOS_LINK.EXE

Catalog of windows used by the Simulation Linker executable.

SIMSET.CAT

Location: C:\HOSIV

Used by: SIMSET.EXE

Catalog of windows used by the Simulation Setup executable.

SIMULA.CAT

Location: C:\HOSIV

Used by: SIM.EXE

Catalog of windows used by the Simulation executable.

VIEWRSLT.CAT

Location: C:\HOSIV

Used by: VIEWRSLT.EXE

Catalog of windows used by the View Results executable.

DATA FILES

HOSSYS.DAT

Location: C:\HOSIV

Created by: SETUPSYS.EXE

Used by: SIM.EXE

HOS system setup parameters

Mass storage device: EXTERNAL for Bernoulli boxes; INTERNAL for internal hard disks.

Integer loop counter for controlling screen scrolls: 100 for bus mouse; 1 for serial mouse.

SIMNAME.DA1

Location: C:\HOSIV\SIMNAME

Created by: Simset.exe

Used by: HOS_LINK.EXE

C language include file for SIM.C containing basic parameters for simulation.

TIME_UNITS = UNIT; where unit is one of the following units: thousandths, hundredths, tenths, seconds, minutes, hours, days.

MAX_SIM_TIME = n; where n is a 6 character long unsigned indicating the maximum simulation time in terms of the minimum time unit.

strcpy (SIM_DESCRIPTION, "xxxxx"); where xxxxx is a maximum of 80 characters containing simulation description.

strcpy (SIM_NAME, "xxxxxxxx"); where xxxxxxxx is the 8 character simulation name.

start_sim_func = function_name; where function_name is the name of the action that is to be invoked at the start of the simulation.

SIM_START_TIME = n; where n is a long unsigned integer containing the simulation start time.

SIMNAME.DA2

Location: C:\HOSIV\SIMNAME

Created by: Simset.exe

Used by: HOS_LINK.EXE

Contains basic simulation information.
xxxxxxxx where xxxxxxxx is the eight-character simulation name.

n where n is an integer (0-6) indicating the minimum time unit as follows:
0=thousandths
1=hundredths
2=tenths
3=seconds
4=minutes
5=hours
6=days

80 character simulation description.

Date and time that simulation was last executed from SIM.EXE.

SIMNAME.DA3

Location: C:\HOSIV\SIMNAME

Created by: Simset.exe

Used by: HOS_LINK.EXE

Contains basic simulation information.
Simulation start time in minimum time units.

Actual date and time that the simulation started running.

Actual date and time that the simulation stopped running.

Date and time that simulation was last executed from SIM.EXE.

ALL_SIMS.DAT

Location: C:\HOSIV

Created by: HOSIV.EXE

Used by: HOSIV.EXE

List of all simulations that have been created.

```
char   sim_name[8];
```

HOSERROR.ERR

Location: C:\HOSIV

Created by: All Modules

Used by: All Modules

Used by all modules to report system related errors, like failure on file access, and memory usage.

OUT.OUT

Location: C:\HOSIV

Created by: Any executable

Used by: All modules

Redirected output from all executables in HOS.

HOSIV.LIB

Location: C:\HOSIV

Created by: N/A

Used by: EDIT_TSK.EXE; EDIT_OBJ.EXE; HPL.EXE

Data base procedures. Runtime library of calls for maintaining the Object data base, procedure and set libraries, and alphabetics dictionary.

Includes the following object modules:

setlib.obj -- set library creation and maintenance procedures.

lowlevdb.obj -- object data base access routines.

objdir.obj -- object directory routines that provides search.

capability for object names.

objset.obj -- runtime set manipulations used by HAL procedures.

dumpout.obj -- formats and prints the object data base.

hosproc.obj -- creates and maintains the action library data base.

objprop.obj -- maintains and uses the alphabetics dictionary.

library.obj -- the routines which are the translation result of the Hal translator. These are the simulation runtime routines for object data base manipulation.

HOSOBJ.H

Location: HOSIV

Created by: LORNA

Used by: HOSIV.LIB

Used only for compilation. HOSIV library header file. Used by all code modules included in the library. Needed only if recompiling one of the HOS executable modules.

HOSPROC.P$

Location: C:\HOSIV

Created by: Action Editor

Used by: HOSIV.LIB and most of the editors including the HOS_LINK process.

Action library data base. Library where the actions and their corresponding files (pnnnnnnnn.[in],[c],[hpl]) are stored.

```
typedef struct {
    char name[31]; /*procedure name*/
    char file[9]; /* filename (pnnnnnnn.??? */
    int status;/* deletion status */
    } PROCLIB;/* procedure library record structure
```

HOSPROP.P$

Location: C:\HOSIV

Created by: Object Editor

Used by: HOSIV.LIB/ and user created simulations as well as the HOS_LINK process.

Alphabetic dictionary.

31 character records, nonvariable length records written to a binary file. Can use random access within the file for reading and writing.

HOSSET.S$

Location: C:\HOSIV

Created by:

Used by: HOSIV.LIB

Object set library data file. Contains all the set names defined, and the temporary file name to used during execution.

```
typedef struct {
    char name[31]; /*set name*/
    char file[9]; /* temporary file name snnnnnnn*/
    int status; /*deletion status*/
    int members; /*number of members in set*/
    } SETLIB;
```

INSTALL.BAT

Location: C:\HOSIV

Created by:

Used by: SETUPSYS.EXE

Installation batch file.

OBJECTS.O$

Location: C:\HOSIV

Created by: Object Editor

Used by: HOSIV.LIB - used be some editors and user created simulations

Object library data file

```
typedef struct{
    char a_name[32]; /*characteristic name*/
    char type; /* characteristic type*/
    union {
        long i; /* storage space for value -- long or double*/
        double f;
        } val;
    } FIELD; /* one field declaration*/
typedef struct {
    char name[32]; /*object name*/
    int retrieved; /*reserved for later use*/
    unsigned int d_status; /*delete status*/
    int set_status; /* set membership indicator*/
    FIELD  f[15]; /*characteristics -- 14 actual, one NIL*/
    } OBJECTS;
```

SIM.C

Location: C:\HOSIV

Created by:

Used by: HOS_LINK.EXE

Portion of the source code required to compile SIM.EXE. Used with SIMULA.C.

This is the actual simulation driver. It processes all the events, rules and associated procedures, as well as outputing all the simulation information.

SIMULA.C

Location: C:\HOSIV

Created by:

Used by: Linked in with sim.c when creating a simulator

Used with SIM.C to compile SIM.EXE.

This is the user interface part of the simulator.

SIMNAME.DEV

Location: C:\HOSIV\SIMNAME\F or F:\

Created by: SIM.C

Used by: SIM.EXE; TIMELINE.EXE and view results

Simulation event data output file.

SIMNAME.DOB

Location: C:\HOSIV\SIMNAME\E or E:\

Created by: SIM.C

Used by: SIM.EXE; TIMELINE.EXE; object analysis and view results

Simulation object data output file.

SIMULATI.EV1

Location: C:\HOSIV

Created by: SIM.EXE

Used by: SIM.EXE

Temporary event data file used in simulation compilation.

SIMNAME.DTK

Location: C:\HOSIV\SIMNAME\F or F:\

Created by: SIM.EXE

Used by: SIM.EXE; TIMELINE.EXE; TASKANAI.EXE and view results

Simulation rule data output file.

SIMNAME.LOG

Location: C:\HOSIV\SIMNAME\F or F:\

Created by: SIM.EXE

Used by: SIM.EXE

Simulation log file created from user-defined FILE statements contained in actions.

SIMULATI.TK2

Location: C:\HOSIV

Created by: Task Editor

Used by: HOSIV.LIB

Actions referenced in rules.

SIMNAME.TK3

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: SIM.C

This file contains the C code to initialize all of the rules. It is a combination of the three files: C:\hosiv\simname.to3, C:\hosiv\simname.te3, and C:\hosiv\simname.th3.

SIMULATI.TK4

Location: C:\HOSIV

Created by: Task Editor

Used by: SIM.C

C code to drive tasks.

SIMULATI.TK5

Location: C:\HOSIV

Created by: Task Editor (task_bool)

Used by: HOS_LINK.EXE

List of all objects and characteristics used in rules.

SIMULATI.TK6

Location: C:\HOSIV

Created by: Task Editor (task_bool)

Used by: HOS_LINK.EXE

List of all alphabetics used in rules.

TASKANAL.RPT

Location: C:\HOSIV\SIMNAME

Created by: TASKANAL.EXE

Used by: View results

Tasks analysis report.

TIMELINE.RPT

Location: C:\HOSIV\SIMNAME

Created by: TIMELINE.EXE

Used by: View results

Timeline report.

SIMNAME.EV1

Location: C:\HOSIV\SIMNAME

Created by: edit_evt.exe

Used by: HOS_LINK.EXE

Event data file.

Each record is in the following format:

xxx_yyyyyy_zzzzzzzzzzzzzzzzzzzzzzzzzzzzzz_a

xxx is three digit event number; yyyyyy is a 6 digit event time zzzzz... is the 31 character event procedure; a... is the 40 character event description.

SIMNAME.EV2

Location: C:\HOSIV\SIMNAME

Created by: edit_evt.exe

Used by: HOS_LINK.EXE

Actions referenced in events.

Every record contains the following format:

zzzzz.

where zzzzz... is a 31 character action referenced in an event.

SIMNAME.EV3

Location: C:\HOSIV\SIMNAME

Created by: edit_evt.exe

Used by: HOS_LINK.EXE

C include file with pointers to event actions.

Every record is in the following format:

event_proc[xxx] = zzzzzzzzz;

where xxx is a three digit event number

zzzzzzzzz... is the action.

SIMNAME.OB

Location: C:\HOSIV\SIMNAME

Created by:

Used by:

SIMNAME.OBS

Location: C:\HOSIV\SIMNAME

Created by:

Used by: HOSIV.LIB

Object library file with object information at conclusion of simulation.

SIMNAME.PRP

Location: C:\HOSIV\SIMNAME

Created by: Action Editor and maybe Task Editor

Used by: HOS_LINK.EXE

Alphabetics referenced in the simulation.

SIMNAME.TK2

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains a list of all procedures referenced in all of the rules. It is a combination of the three files: C:\hosiv\simname.to2, C:\hosiv\simname.te2, and C:\hosiv\simname.th2.

There is one action name per line separated from the next by a carriage return linefeed combination.

SIMNAME.TK4

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains the C code to drive all of the rules. It is a combination of the three files: C:\hosiv\simname.to4, C:\hosiv\simname.te4, and C:\hosiv\simname.th4.

SIMNAME.TE1

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: edit_tsk.exe

This file contains the text version of the environment rules. It is formatted for output to the printer including form feeds.

SIMNAME.TH1

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: edit_tsk.exe

This file contains the text version of the hardware rules. It is formatted for output to the printer including form feeds.

SIMNAME.TO1

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: edit_tsk.exe

This file contains the text version of the operator rules. It is formatted for output to the printer including form feeds.

SIMNAME.TKO

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: edit_tsk.exe

This file is used by the Rule Editor to store its internal representation of the operator rules. The program uses an fwrite to put the structure on the disk once for each rule.

```
char  description[35];
char  if_cond[200];
char  if_c[300];
char  procedure[35];
char  until_cond[200];
char  until_c[300];
char  priority;
char  sub_priority;
task_type    *next;
task_type    *last;
```

SIMNAME.TKH

Location: C:\HOSIM\SIMNAME

Created by: edit_tsk.exe

Used by: edit_tsk.exe

This file is used by the Rule Editor to store its internal representation of the hardware rules. The program uses an fwrite to put the structure on the disk once for each rule.

```
char  description[35];
char  if_cond[200];
char  if_c[3  )];
char  procedure[35];
char  until_cond[200];
char  until_c[300];
char  priority;
char  sub_priority;
task_type   *next;
task_type   *last;
```

SIMNAME.TKE

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: edit_tsk.exe

This file is used by the Rule Editor to store its internal representation of the environment rules. The program uses an fwrite to put the structure on the disk once for each rule.

```
char   description[35];
char   if_cond[200];
char   if_c[300];
char   procedure[35];
char   until_cond[200];
char   until_c[300];
char   priority;
char   sub_priority;
task_type    *next;
task_type    *last;
```

SIMNAME.TOI

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: sim.c

This file contains the C code to initialize the operator tasks.

SIMNAME.TEI

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: sim.c

This file contains the C code to initialize the environment tasks.

SIMNAME.THI

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: sim.c

This file contains the C code to initialize the hardware tasks.

SIMNAME.TOC

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: sim.c

This file contains the C code to execute the operator tasks.

SIMNAME.THC

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: sim.c

This file contains the C code to execute the hardware tasks.

SIMNAME.TEC

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: sim.c

This file contains the C code to execute the environment tasks.

SIMNAME.TO2

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains a list of all procedures referericed in the operator rules.

There is one action name per line separated from the next by a carriage return linefeed combination.

SIMNAME.TE2

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains a list of all procedures referenced in the environment rules.

There is one action name per line separated from the next by a carriage return linefeed combination.

SIMNAME.TH2

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains a list of all procedures referenced in the hardware rules.

There is one action name per line separated from the next by a carriage return linefeed combination.

SIMNAME.TK5

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains the list of all attribute object pairs referenced in the rules. it is a combination of the three files: C:\hosiv\simname.to5, C:\hosiv\simname.te5, and C:\hosiv\simname.th5.

attribute, object

The attribute is separated from the object by a comma and a space. Each pair is separated by a carriage return linefeed combination.

SIMNAME.TO5

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains the list of all attribute object pairs referenced in the operator rules.

attribute, object

The attribute is separated from the object by a comma and a space. Each pair is separated by a carriage return linefeed combination.

SIMNAME.TE5

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains the list of all attribute object pairs referenced in the environment rules.

attribute, object

The attribute is separated from the object by a comma and a space. Each pair is separated by a carriage return linefeed combination.

SIMNAME.TH5

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains the list of all attribute object pairs referenced in the hardware rules.

attribute, object

The attribute is separated from the object by a comma and a space. Each pair is separated by a carriage return linefeed combination.

SIMNAME.TK6

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains the list of all alphabetics referenced in all of the rules. It is a combination of the three files : C:\hosiv\simname.to6, C:\hosiv\simname.te6, and C:\hosiv\simname.th6.

alphabetic

Alphabetics are separated by carriage return linefeed combinations.

SIMNAME.TO6

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains the list of all alphabetics referenced in the operator rules.

alphabetic

Alphabetics are separated by carriage return linefeed combinations.

SIMNAME.TE6

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains the list of all alphabetics referenced in the environment rules.

alphabetic

Alphabetics are separated by carriage return linefeed combinations.

SIMNAME.TH6

Location: C:\HOSIV\SIMNAME

Created by: edit_tsk.exe

Used by: HOS_LINK.EXE

This file contains the list of all alphabetics referenced in the hardware rules.

alphabetic

Alphabetics are separated by carriage return linefeed combinations.

SIMNAME.DA4

Location: C:\HOSIV\SIMNAME

Created by: simset.exe

Used by: HosLink.exe

Contains the simulation start action.

Simulation start action, 31 character maximum.

SEGLOAD.DAT

Location: C:\HOSIV

Created by: VIEWRSLT.EXE

Used by: OBJANAL.C

Temporary file used to pass an object name to the object analysis program.

char name[32]; /*name of the object to look for*/